MANCHESTER 1824

The University of Manchester

STRUCTURAL COMPLEXITY IN MUSIC MODELLING AND GENERATION WITH DEEP NEURAL NETWORKS

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE SCHOOL OF ENGINEERING

2021

By
Jacopo de Berardinis
Department of Computer Science

Contents

Word Count: 104344

Abstract

STRUCTURAL COMPLEXITY IN
MUSIC MODELLING AND GENERATION
WITH DEEP NEURAL NETWORKS

Jacopo de Berardinis

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy, 2021

The automatic composition of music is a historical challenge that dates back to the ancient Greeks, and remained up to and beyond Mozart with the "Dice Game" and Ada Lovelace, speculating that the "calculating engine" might compose elaborate and scientific pieces of music of any degree of complexity or extent. In recent years, data-driven generative systems based on deep learning architectures have achieved impressive results on symbolic music, and they can also produce acoustically realistic outputs when trained on the raw audio. Nevertheless, composing musical ideas longer than motifs and phrases is still an open challenge in computer-generated music, a problem that is commonly referred to as the lack of long-term structure in the generations. In addition, the evaluation of the structural complexity of artificial compositions is still done manually – requiring expert knowledge, time and involving subjectivity which is inherent in the perception of musical structure.

The thesis addresses this specific research gap by introducing a collection of methods and tools for the automatic evaluation of structural complexity in music. First, a music modelling framework is introduced, allowing experimenters to design, train and evaluate their music models, and compare their performance with state of the art methods. This is fundamental considering that the structural problem stems from the notorious technical challenges of the task on which these generative systems are trained – the problem of learning long-term dependencies in sequential data. Second, to evaluate structural complexity from arbitrary music pieces, a novel algorithm for music structure analysis and a set of metrics were designed. The former is used to segment music hierarchically by detecting nested structural segments at all possible temporal levels. This is followed by the extraction of a set metrics to formally describe the decomposition process of music into its structural components. The algorithm was found to

achieve state-of-the-art performance in hierarchical music structure analysis, and the structural metrics succeeded in distinguishing music belonging to different structural complexity groups. Finally, the thesis concludes with an investigation on the use of structural properties of music to potentially improve the performance and the interpretability of methods for music information retrieval, with a case study in music emotion recognition.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

Mentioning all people who contributed to this work, implicitly or explicitly, and to my professional development, intentionally or unintentionally, would probably take more than a few pages. As I prefer not to take the risk of forgetting any name, I **`thank you all`** in the humblest and sincerest way. In particular, I am extremely grateful to three key persons, in order of appearance during my doctoral studies: Dr. Ke Chen, for offering me the opportunity to work together; Prof. Angelo Cangelosi, for having believed in me in the first place, and also been my "academic father" throughout the whole journey; and finally, Dr. Eduardo Coutinho for having thought me so much and for his remarkable patience, temper and support. A special thanks also goes to Dr. Gabriella Pizzuto, who provided constructive feedback on the manuscript, and gave me some English lessons for every sentence she corrected and/or improved.

Part I

Opening

Chapter 1

Introduction

Structure is at the basis of human expression and communication, serving as a backbone to organise content and relate information. Just by looking at the index of this thesis, it is evident that all its content is organised in four parts, each of which comprises a number of chapters contextually related to each other. In turn, the structure of a chapter depends on its function within the part it belongs to, and includes sections, subsections, paragraphs, sentences, and further nested material until finding words – the basic building blocks.

Similarly to other human-created content, the structural layout of language is both functional and hierarchical, meaning that verbal structures carry a precise meaning and exhibit a nested organisation. Music makes no exception, and presents additional properties that further increase the complexity of the structural layout, such as harmony, polyphony – the interposition of multiple voices, instrumentation, and expressive variations – either explicit in the notation, or resulting from the performer's interpretation of a piece. Although all these properties are naturally recognised by humans, the structural complexity of music represents an obstacle for any computational procedure analysing music signals.

1.1 Order and chaos: structural properties of music

Music is a powerful medium that conveys meaning to listeners by combining a variety of musical elements synchronously and sequentially. At the perceptual level, the basic attributes involved in music perception are loudness, pitch, contour, rhythm, tempo, timbre, spatial location and reverberation (Levitin, 2006). Whilst listening to music, our brains continuously track and analyse these signals according to diverse gestalt and psychological schemas. Some of them entail higher order musical dimensions (e.g., metre, key, melody, harmony), which reflect (contextual) hierarchies, intervals and regularities between the different musical elements. Others involve continuous predictions about what will come next in the music as means of tracking

structure and conveying meaning (Meyer, 2008).

Structural elements of music can range from local/short-term organisational levels (e.g., chord, a sequence of notes/sounds) – the "micro" level – to the longer temporal scales capturing the form of a composition or compositions (e.g., sonata form in classical music, or verse/chorus form in popular music) – the "macro" structure. Within these levels, patterns can be identified and music can be segmented in various ways on the basis of specific musical characteristics at different temporal levels (e.g., dynamics, patterns of durations/rhythm, melodic patterns, instrumentation, etc.) (Lerdahl, Jackendoff, & Jackendoff, 1983).

Given that the same musical material may induce structure at different interrelated levels, one interesting feature of musical organisation is its hierarchical nature. For instance, a piece of music may be analysed in terms of its overall form (e.g., divided into meaningful sections), but within those sections we can further divide music into sub-levels that reflect, for instance, rhythmic or harmonic structure. Naturally, given the diversity of musical styles and compositional/performative approaches, different pieces/performances will have different kinds and amounts of (hierarchical) structure, and therefore diverge in terms of structural complexity[1].

Indeed, one of the most complex aspects of music is the way a composer organises and relates his or her musical ideas within a composition. Different musical ideas are in general presented to provide contrast and surprise the listener, whereas others are repeated at different times or even varied in order to create a sense of familiarity. Not only are these musical patterns closely inter-related, but they can also be decomposed into progressively shorter ideas in light of their hierarchical organisation. This interplay between similarity, novelty and hierarchical decomposition of musical patterns is coherently organised in such a way as to convey the composer's unitary vision of the piece. This structural aspect of notated and performed music is known as musical form, which can be defined as a genre specific definition of the expectation of how a piece is composed at different time scales. Musical form is indeed one of the most evident examples of the latent hierarchical structures encoded in music. In Western music, at the most granular level it consists of individual notes and chords – the basic building blocks of a composition; by combining them sequentially and synchronously we obtain larger structural constructs such as measures, motives and phrases, which in turn contribute to the definition of sections. Examples of sections in popular music are *intro*, *chorus*, and *verse*, whereas in classical music we can find *exposition*, *development* and the *recapitulation* of a movement (Müller, 2015). These high-level structural elements determine the overall layout of a composition, and the resulting nested organisation of sounds makes it possible to visualise the hierarchical structure of a piece using tree representations of music (Lerdahl & Jackendoff, 1985).

[1]For a detailed perspective on the theoretical analysis of music structure, we refer to (Goetschius, 1904), and to (Lerdahl et al., 1983) or (Meyer, 2008) for a computational treatment of the subject.

The varied and sophisticated patterns of structure that characterise music are a key distinguishing factor when compared to other acoustic mediums (e.g., speech, soundscapes). In fact, the importance of music structure to musical appreciation is paramount (Gaver & Mandler, 1987) and a wide range of musical parameters as well as structural features are fundamental to covey different types of meaning to listeners (Patel, 2003), which in turn can trigger a cascade of other responses (e.g., dancing, emotions) (Gomez & Danuser, 2007).

1.2 The problem of learning musical structure

Due to its intrinsic complexity, music is a sequential process that is particularly compelling to analyse computationally. Indeed, new methods for temporal information processing are often tested on music signals, and this is quite common for all those methods and paradigms that are generally applicable to *sequence modelling*. In this context, autoregressive (AR) models are trained to process a sequential stream – one step/event at a time, in order to predict the next event in the sequence. From a general perspective, a sequence model capable of capturing the underlying distribution to which the sequential data is subject, is a powerful computational tool that can support several tasks (e.g. sequence prediction, sequence generation), and help gaining a deeper understanding of the phenomenon under study, such as the discovery of non-trivial relationships and patterns among features and events.

When the sequential data is music, the problem is commonly known as *music modelling* – a general task enabling a number of applications, from automatic music transcription and style transfer, to machine improvisation and automatic music composition (c.f. Section 2.1.2). Among all the musical properties contributing to the complexity of this problem, structure is one of the most challenging for a music model. More precisely, any algorithmic procedure for music modelling has to deal with the structural texture of a composition/performance – the presence of nested and interrelated structural segments reflecting the organisation of the musical ideas conceived by the composer and/or emphasised by the performer. For a music model, this necessitates relating musical content at different time scales, while recognising if any relationships of repetition, variation, and contrast are implied among them. As pointed out by Eck and Schmidhuber (2002a) such complex interactions require memorisation capabilities. In particular, a model should implement a content-addressable memory to store intermediate representations of musical content whenever a potential structural segment is detected. This way, when a model is processing a repeated segment, say a phrase, it would leverage its memory to retrieve the corresponding representations needed to replicate all the notes in that phrase.

Despite the recent progress in the field, relating musical content at the coarsest time scales (e.g. at the sectional level) is still an open problem in music modelling. Therefore, although

state of the art (SOTA) music models can succeed at retaining short-term structures, such as figures and motives, modelling larger segments like sections and movements is currently beyond their capabilities. This is known as the challenge of learning long-term structure in music, and is directly linked to the well-known problem of learning long-term dependencies from sequential data (Bengio, Simard, & Frasconi, 1994) – a long-standing goal in machine learning research.

1.3 Motivations and implications

The problem of learning structural properties of music is particularly evident when a model is trained on musical corpora to generate new compositions. Not only the model will struggle to predict those events belonging to repeated or varied long-term structures, but the musical material generated from the model will consequently reflect this limitation. In other words, as discussed later, a model struggling to learn long-term structure cannot be expected to generate music with a realistic level of structural complexity. In addition, and to our best, there is no automatic method or criteria to objectively measure the structural capabilities of a music model, neither after the training stage nor at generation time.

1.3.1 An open challenge for automatic music composition (AMC)

In the last few years, composing music with machine learning (ML) systems has attracted great interest from academia and industry (Fiebrink, Caramiaux, Dean, & McLean, 2016). Companies started offering AMC solutions for entertainment content, such as soundtracks for video games and commercials. Researchers, instead, are leaning towards *computer-assisted composition*, augmenting the creative potential of artists and composers (Papadopoulos, Roy, & Pachet, 2016); and *machine improvisation*, a category of intelligent systems capable to temporarily replace a performer during a live session (Martin, Ellefsen, & Torresen, 2018). Improving the generative capabilities of these systems does not only opens up the investigation of new forms of music, but is also considered a pinnacle to understand machine creativity (Brunner, Wang, Wattenhofer, & Wiesendanger, 2017).

Nonetheless, dealing with the structural complexity of music has been a tremendous challenge, especially for generating long and musically meaningful pieces endowed with form and long-term structure (Briot, Hadjeres, & Pachet, 2020; Herremans, Chuan, & Chew, 2017). Indeed, current SOTA systems generate pieces that are mostly characterised by local or short-term form, with motives – the shortest musical ideas, dominating the synthetic compositions (Eck & Schmidhuber, 2002b). This is particularly prominent in music generated from long-short term memory (LSTM) recurrent neural networks (RNNs) (Hochreiter & Schmidhuber, 1997).

The advent of self-attention networks (SAN) in music modelling ameliorated this problem (Vaswani et al., 2017), with Transformer models now capable of generating music possessing structural properties that remain more coherent across a larger temporal scale compared to LSTMs (C.-Z. A. Huang et al., 2018). Nevertheless, there is general consensus on the fact that the automatic generation of music with a realistic level of structural complexity is still an open problem for most genres. In fact, although structures at different temporal scales can now be found in generated music, those are rarely organised to convey a coherent musical idea throughout the piece, and inter-related with each other based on the principles of *repetition*, *variation* and *contrast* (Müller, 2015).

Compared to the symbolic domain, the problem of structural complexity is more challenging for *audio waveform generation*, as it requires processing significantly longer sequences (a three-minute-long audio segment sampled at 44.1 kHz will have an input length of about 8 million time steps). Not only this exacerbates the problem of learning long-term dependencies, but generative models of waveform also have to capture an additional wide range of musical properties (e.g. timbre). In the audio domain, autoregressive models have been demonstrated to model local signal variations effectively and capture temporal correlations across tens of seconds (Dieleman, Oord, & Simonyan, 2018). A recent state-of-the-art AMC system is JukeBox (Dhariwal et al., 2020) – generating audio music conditioned on artist, style, and lyrics. This model, counting billions of (learnable) parameters, was trained for several weeks using more than 512 V100 GPUs. Nevertheless, when describing the generated musical repertoire, the authors reported that they could not "*hear long term musical patterns, and [...] choruses or melodies that repeat*" (Dhariwal et al., 2020). Analogously, when using JukeBox to generate completions close to the original pieces, they found that the generated continuations would "*deviate completely into new musical material after about 30 seconds*".

1.3.2 Evaluation of the structural capabilities of AMC methods

In our view, to start tackling this problem it is necessary to evaluate the structural complexity of music generated by AMC systems, in order to have a reference point that can be used to improve their composition capabilities. However, the evaluation of music generation methods is another open issue in the field, considering the lack of a standard evaluation methodology that can enable and foster a fair and objective comparison of AMC systems on a large scale (L.-C. Yang & Lerch, 2020).

In fact, evaluation is always demanded when submitting a novel music generation method. Nonetheless, different evaluation criteria and strategies are used heterogeneously and in isolation from each other. In most cases, evaluation relies on manual and subjective judgements by human listeners whom provide ratings on specific properties related to the music composition

itself (e.g. pitch range, mode, rhythmical consistency) or their subjective evaluation of the listening experience (e.g., likeability, originality). In some (rare) cases, expert listeners are asked to evaluate the generated pieces by analysing their musical properties as a music teacher would do with the composition of a student (B. L. Sturm & Ben-Tal, 2017).

Overall, in line with the taxonomies reported in (Carnovalini & Rodà, 2020) and (L.-C. Yang & Lerch, 2020), evaluation methods for music generation can be organised into the following categories – rarely used in conjunction.

Music modelling evaluation. It concerns the evaluation of the prediction performance of an autoregressive music model – a specific family of music generation systems (also known as *predictive models for music*) that are trained to predict the next musical token (e.g. a note, chord, or a quantised representation of musical material) given the context of the previous events in a musical sequence (analogously to language models). This type of evaluation is based on the assumption that a model that can effectively predict music – having learnt associations between past and future musical content, can potentially encapsulate notions of music perception and composition. Hence, evaluating the predictive capabilities of a music model provides an indicator of the learned musical features possibly reflecting theoretical properties of music. The most common quantitative evaluation measures in the literature are the *log-likelihood* of the model's predictions on the test set, *frame-level accuracy* (Boulanger-Lewandowski, Bengio, & Vincent, 2012), as well as general *classification measures* such as F-measure, precision, recall and perplexity (Ycart, Benetos, et al., 2017).

Statistical comparisons. Methods belonging to this category are based on computing some descriptive statistics on a set of generated compositions so that they can be compared with those extracted from the training data. Examples of these statistics at the piece level are *pitch and note counts*, *pitch class and note length histograms*, *average pitch interval* and so forth. Hence, this comparison provides a weak measure of the resemblance of the generated sequences to those contained in the training set (L.-C. Yang & Lerch, 2020), which can also be interpreted as a "plagiarism score" in a way (B. L. Sturm, Santos, Ben-Tal, & Korshunova, 2016). Nonetheless, a high level of similarity with the training material might also indicate an overfitting trend or a poorly configured sampling strategy.

Composition evaluation. The purpose of this evaluation is to formally assess the quality and the plausibility of generated pieces in terms of musical properties and/or theoretical rules. This can be done via computational measures derived from musicologists methods (Chuan & Herremans, 2018), or by involving a community of music experts for review (B. L. Sturm & Ben-Tal, 2017). Given the scarcity of computational measures that can automate this process, the manual evaluation of generated compositions, on the other hand, is a laborious task requiring a

high level of musical expertise. In addition to potentially not being accessible, this evaluation methodology also involves subjectivity at different levels.

Listening tests. This last group collects two of the most common evaluation methods found in music generation works. Both these strategies are based on listening tests involving human participants, often without any musical training.

Turing test (alias *discrimination test*). A group of listeners with different musical background is presented with pieces either composed by humans or generated by a model. Listeners are asked to discriminate among these groups, which basically corresponds to answering the question: *was this piece composed by a human or by a machine?* Whereas a model generating music that cannot be clearly distinguished from human work is a positive indicator of its generative capabilities, this "pass-or-fail" methodology does not allow comparisons with other AMC systems. Furthermore, Turing tests have been heavily criticised over the past decades (Ariza, 2009; Pease & Colton, 2011), particularly due to the complex design of listening experiments under these settings (L.-C. Yang & Lerch, 2020).

Blind comparison. This methodology permits to compare music generated from different systems (usually a very few pieces per model under analysis) by letting listeners rate compositions based on specific properties, or express a preference among a given music selection including one piece from each system. The final goal is to measure the extent to which each generated track shows certain properties that would be expected from real compositions. This approach thus provides an evaluation method that allows the ranking of each model according to the so obtained measurements. From a critical perspective, this methodology is sensitive to potential biases emerging from the selection of tracks in the experiment. The latter is particularly concerning in light of the limited collections under analysis.

To the best of our knowledge, most works based on listening tests rely on *crowd-sourcing platforms*, where participants receive a fee for their evaluation (e.g. Amazon Mechanical Turk); or on *web-based platforms* anyone can access to contribute their feedback. Hence, these experiments should be carefully designed (L.-C. Yang & Lerch, 2020), as ensuring an adequate level of control can be challenging considering that participants may not be easily filtered with a desired degree of specificity.

In conclusion, there is a lack of systematic and standardised methods for evaluation of the music generated by AMC systems, which is a major limitation in this area given that there is still no consensus on how music generated from different models can be evaluated and compared.

1.4 Research questions

As motivated in the previous sections, this thesis addresses a specific research gap – the automatic evaluation of computational methods for music modelling and generation in regard to their ability to (i) learn structural properties from symbolic music and (ii) generate compositions endowed with a realistic level of structural complexity, respectively. The former is a prerequisite to the latter, meaning that a model failing at modelling music in a way to relate structural components at all the possible time scales, cannot be expected to generate music with a realistic level of structural complexity. Therefore, both the modelling and the generation problems – stemming from the major challenge of learning long-term dependencies in sequential data, should be considered together when evaluating a music model. This consideration provides the basis of the first two research questions that are specifically addressed in this thesis.

This first research question focuses on the modelling task, as most methods for AMC – before being used for generation, are trained to maximise a learning objective (e.g. minimising a loss function), which is inherently formulated in a way to let the model learn musical properties and association rules (comprising structural relationships). Evaluating a music model on the specific learning task it was trained on thus provides a first indication of how effectively the model managed to capture musical features and composition rules, rather than attempting to memorise the training sequences (an overfitting behaviour which can be detected with due care). Prior to the assessment of the compositions that a model can generate (by sampling from the learned distribution), its modelling capabilities are thus evaluated on a test set.

RQ 1. *How do different modelling paradigms and neural network architectures influence the performance of a music model learning structural properties from music?*

When designing a sequence model for symbolic music, several modelling paradigms and architectural properties are available to the modeller and little is known about how these design choices affect the performance of the resulting model and the type of features it can learn. These modelling choices can also be sensitive to the way symbolic music is represented before being fed to the network (music encodings) and their suitability may depend on the music collection under analysis, according to its size, the type and complexity of the music it contains, the duration of each piece and so forth. Therefore, the first research question aims at obtaining more insights on the impact and the suitability of the main design alternatives for music modelling, thereby enabling a more informed evaluation of the various architectural components.

Once the former type of evaluation is performed, the next step is to assess a repertoire of synthetic compositions generated from the same model. Although a comprehensive evaluation of computer-generated music should touch a number of musical properties, this work focuses on the structural complexity of music. As motivated in Section 1.3, and to the best of our

knowledge, this kind of evaluation is currently performed through manual inspection of the compositions by a music expert. Despite the detailed expert feedback it can potentially provide, this methodology requires considerable time and efforts, hence it is usually operated on a limited set of compositions. In addition, the subjectivity of human judgement may discourage the comparability of different evaluations, meaning that the only way to ameliorate this issue with the same methodology would be to involve multiple experts. Trivially, this would further exacerbate the former concerns and might not be viable or even accessible to the experimenter. In light of these considerations, the second research question is formulated as follows.

RQ 2. *How is it possible to formalise and measure the structural complexity of music in order to enable the automatic evaluation of AMC systems from a given set of generated compositions?*

With this research question, I seek to investigate whether the evaluation of structural complexity can be automatised via an algorithmic procedure applied to generated music. Not only this computational approach would allow a quick and objective evaluation of large music collections, but it would also be easily accessible to all practitioners regardless of their musical background. Overall, a solution to RQ 2 would provide a fast, accessible and consistent computational method enabling the evaluation and the comparison of AMC systems on the structural complexity of their generations. Combined with the previous research question, we aim at establishing a systematic approach to evaluate a music model after training and after generation.

Nevertheless, RQ 2 is intentionally of general scope, as it seeks to address the evaluation of structural complexity from arbitrary music pieces – regardless of the underlying generation process (e.g. autoregressive music models, rule-based systems, semi-automatic methods, template-filling algorithms, music in-painting, etc.) and their format. In other words, an algorithm quantifying structural complexity would only look at the final product(s) of an AMC system, and should be effective for both audio and symbolic music.

Considering the specific focus on the structural properties of music, the next research question intends to investigate if any relationship can be found with other tasks in music information retrieval (MIR). More precisely, as the role of structure has been demonstrated to be linked with triggering emotional responses to listeners (Livingstone, Palmer, & Schubert, 2012; Sloboda, 1991), the main quest revolves around the use of structural information to improve the performance and the interpretability of music emotion recognition systems. These structural cues can be related to the formalisation of structural complexity emerging from RQ2 (as structural features) or interest other structural properties of music besides form (e.g. instrumentation).

RQ 3. *Could we improve the interpretability and the performance of music emotion recognition (a specific task in MIR) by leveraging the availability of music structural properties?*

This last research question concludes the investigation and aims to shed light on the potential impact of this research in other application domains besides music modelling and generation. Compared to the former research questions, RQ 3 has a more investigative nature that is useful to contextualise the main outcomes of this thesis – what can be achieved from addressing RQ1 and RQ2, in regard to their transferability when opening new research directions in the field.

1.5 Contributions to knowledge

The main contributions of this thesis, in relation to the research questions presented in the previous section, are schematically illustrated in Figure 1.1 and outlined as follows.

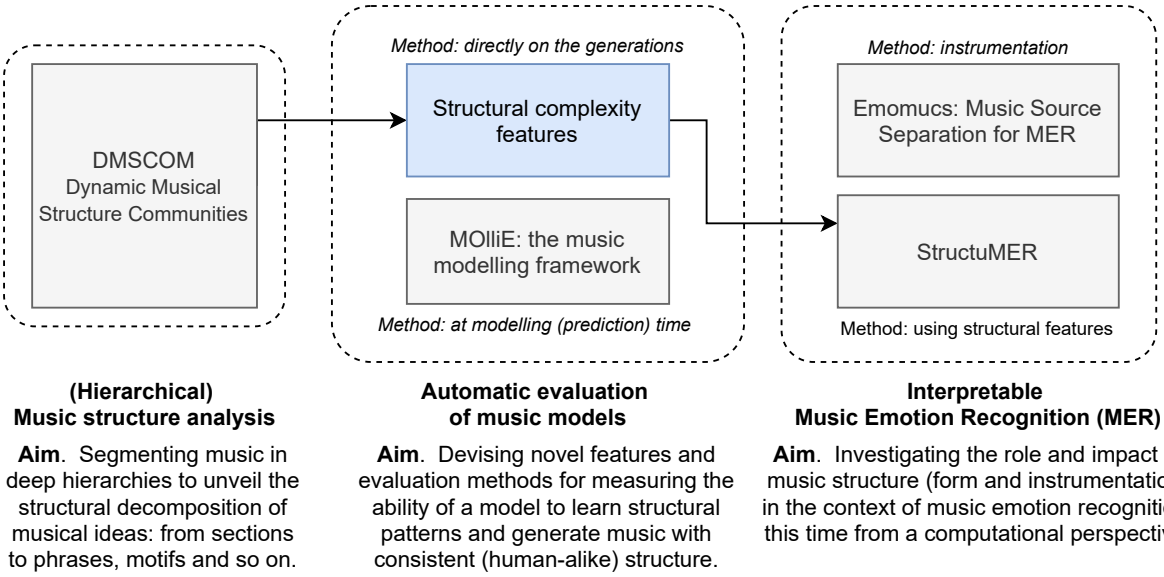


Figure 1.1: Schematic overview of the main contributions.

1.5.1 Evaluating autoregressive models for music

To address RQ1, the first music modelling framework (MOlliE) for symbolic music was introduced. Our framework provides a collection of state-of-art neural network architectures, symbolic music encodings, and visualisation utilities allowing experimenters to design, evaluate and compare their music models in a controlled environment. MOlliE focuses on a specific family of computational models – autoregressive (AR) models for music (alias predictive models or sequence models for music), which are trained to predict the next event in a sequence given the historical context of the previous events. In music modelling, events (or tokens) are notes and silences, temporal slices of notes, or groupings of notes, depending on the granularity of the

chosen music representations. Thanks to this framework, different modelling paradigms, representations of symbolic music, and training procedures can be tested on a number of datasets, allowing the evaluation and the comparison of novel AR models for music with the SoA.

In addition, discovering novel architectures for music modelling that can effectively learn structural dependencies at all possible levels, would directly benefit research in music generation. Indeed, as motivated in Section 1.2, a model that has learned structural features from music, is more likely to generate structurally richer compositions at inference time.

1.5.2 Measuring the structural complexity of music

Addressing RQ2 required the design of two computational methods for the automatic analysis of music structure and the subsequent extraction of structural complexity indicators. Therefore, the second contribution is a novel algorithm for hierarchical music structure analysis (MSA) – an automatic procedure segmenting music at multiple resolution levels in order to detect structural patterns of variable length and complexity. Our method, named musical structure communities (MSCOM), addresses the music segmentation task as a community detection problem and achieved state of the art performance on a well-known benchmark dataset. The key strengths of MSCOM are its unsupervised nature and the unconstrained algorithmic formulation, enabling the detection of structures of all types and at all levels.

The automatic detection of musical structure in audio recordings is one of the most challenging problems in the field of music information retrieval, since even human experts tend to disagree on the structural decomposition of a piece of music. However, most of the current music segmentation algorithms in literature can only produce flat segmentations, meaning that they cannot segment music at different levels in order to reveal its hierarchical structure. This is indeed a crucial limitation considering RQ2, as the definition of structural complexity should take into account all the nested and articulated musical ideas that can be found from a piece.

As MSCOM can segment music hierarchically, the third contribution leverages the rich, unbounded, and informative hierarchies it produces to further summarise their structural properties. More precisely, we introduced a set of metrics that can formally describe the decomposition process of the musical ideas (structures) detected in a piece using MSCOM. Hence, in lieu of subjectively defining structural complexity, it is hypothesised that the former is a latent property that can be captured by a set of metrics compactly describing structural segmentations. This hypothesis was confirmed by an experimental evaluation, as such metrics were found to quantify structural properties of music and to distinguish music belonging to different structural complexity classes. In addition, a simple yet effective evaluation framework was designed to use these metrics for the automatic analysis of structural complexity from music. By automating this process in a deterministic, objective, and efficient way, researchers in music generation

can now use the contributed framework to evaluate the output of their models and compare, at a large scale, the returned structural complexity with that of other state of the art systems.

1.5.3 Investigating structure in music emotion recognition

To address the last research question, we investigated the use of different structural properties of music for the task of music emotion recognition (MER). This was done in two different ways, depending on the type of structural information that was used as input to MER models. First, we used the structural complexity metrics derived from the previous contribution. From our results, we found that some of them are retained by feature selection methods and were found to be useful for predicting induced emotions (valence and arousal regression).

Secondly, we experimented with another structural property of music – the instrumentation of a musical piece. This resulted in the introduction of a novel computational model (EmoMucs) that considers the role of different musical voices in the prediction of the emotions induced by music. EmoMucs combines source separation algorithms for breaking up music signals into independent song elements (vocals, bass, drums, other) and end-to-end machine learning techniques for feature extraction and emotion modelling. Compared to other baselines, our results demonstrated that this method achieves better performance for valence recognition, and comparable ones for arousal, while providing increased interpretability.

In sum, our results demonstrated that structural properties of music related to music form and instrumentation could play a role in the way music triggers emotions to listeners.

List of publications

A list of publications highlighting the contributions of this thesis are provided below. In these works, Cangelosi and Coutinho provided supervisory roles and feedback on the manuscripts. As for the other co-authors, Vamvakaris implemented the community detection part of the MSCOM algorithm; and Barrett designed and implemented the neural networks architectures provided in MOlliE. Therefore, the use of the first plural person in this thesis (we/our) is intended to give credit to all the co-authors who contributed in the aforementioned way.

- **Jacopo de Berardinis**, Angelo Cangelosi, Eduardo Coutinho, “Measuring the structural complexity of music: from structural analysis to the automatic evaluation of models for music generation” in the IEEE/ACM Transactions on Audio, Speech, and Language Processing (manuscript accepted).
- **Jacopo de Berardinis**, Michail Vamvakaris, Angelo Cangelosi, Eduardo Coutinho, “Unveiling the Hierarchical Structure of Music by Multi-Resolution Community Detection”

in Transaction of the International Society for Music Information Retrieval 3(1).

- **Jacopo de Berardinis**, Samuel Barrett, Angelo Cangelosi, Eduardo Coutinho, "Modelling long- and short-term structure in symbolic music with attention and recurrence" in Proceedings of the 2020 Joint Conference on AI Music Creativity, Stockholm, (Sweden), October 2020.

- **Jacopo de Berardinis**, Angelo Cangelosi, Eduardo Coutinho, "The Multiple Voices of Musical Emotions: Source Separation for Improving Music Emotion Recognition Models and their Interpretability" in International Society for Music Information Retrieval (ISMIR), Montreal (Canada), October 2020.

Other publications produced during the period of this thesis, although not directly related to the doctoral project are reported below.

- **Jacopo de Berardinis**, Gabriella Pizzuto, Francesco Lanza, Jorge Meira, Antonio Chella, Angelo Cangelosi, "At Your Service: Coffee Beans Recommendation From a Robot Assistant" in ACM Proceedings of Human Agent Interaction (HAI), Sydney (Australia), November 2020.

- Flavio Corradini, Francesco De Angelis, **Jacopo de Berardinis**, Carlo Castagnari, Giorgio Forcina, Andrea Polini, "Tangramob: an agent-based simulation framework for validating smart mobility solutions" in Journal of Intelligent Systems.

- **Jacopo de Berardinis**, Giorgio Forcina, Ali Jafari, Marjan Sirjani, "Actor-Based Macroscopic Modeling and Simulation for Smart Urban Planning" in Science of Computer Programming.

1.6 Structure of the thesis

The structure of this document is illustrated in Figure 1.2 in order to simplify its consultation and highlight the main functional dependencies between chapters. Following this introduction, the main parts of the thesis are background and contributions.

The background part is composed of three chapters, which are presented as follows.

Chapter 2 introduces the preliminary concepts behind the music modelling task. In particular, it presents the mathematical formulation of the problem and its main variants when different musical properties are progressively taken into account (polyphony, multiple parts). The analogies with the language modelling task in natural language processing are also outlined to emphasise the increased complexity of modelling musical sequences compared to textual data.

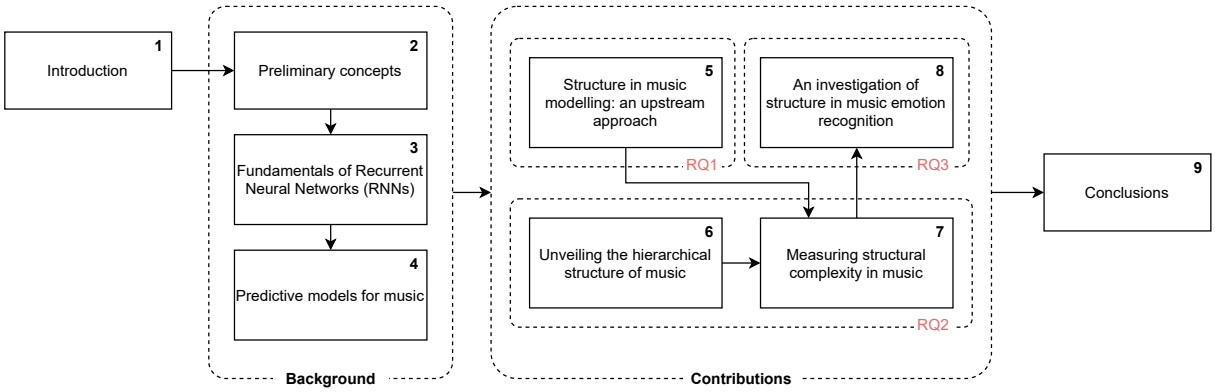


Figure 1.2: The high-level organisation of the thesis. An arrow from one chapter to another indicates that the former chapter is prerequisite material for understanding the latter.

This is further complemented with an overview of potential applications that can benefit from the use of music models, including automatic composition. Finally, the chapter provides an overview of the other tasks in MIR that are relevant to this project, namely music structure analysis (MSA) and music emotion recognition (MER). As motivated in the previous section, the former is a fundamental part of the methodology to address RQ2, whereas the latter represents an application scenario to investigate the influence of structure in other MIR tasks (RQ3).

Chapter 3 provides a compact yet comprehensive overview of artificial neural networks, mainly focused on recurrent architectures that can be used for sequence modelling – the general ML task where music modelling stems from. This also includes architectural contributions and modelling paradigms in order to highlight how ML research has tried to address the problem of learning long-term dependencies in sequential data. Hence, the aim of the chapter is twofold: to introduce the most common recurrent architecture for sequence modelling, and to offer a literature review of ML methods specifically designed to mitigate the problem in question.

Chapter 4 restricts the context of the previous chapter to the specific domain of music modelling – when the sequential data under modelling represents symbolic music. Although the focus of this thesis is on learning long-term structure in music, the chapter provides a detailed overview of the main properties and technical challenges of the music modelling problem. This is followed by a systematic literature review of music models based on a taxonomy relating these works to the technical challenges they intend to address. The chapter concludes by outlining open research gaps in the field in light of the current limitations identified from the literature.

The contribution part is composed of four chapters, following the same order of the contribution section (Section 1.5). Each contribution chapter follows a similar outline: reiterating the main research goals and objectives, in relation to the research questions, detailing the proposed method, and providing an empirical evaluation of the latter on specific music datasets.

Chapter 5 presents the music modelling framework (MOlliE) and the rationale behind its

development and use. We discuss the main modelling paradigms and architectural types found in state of the art music models and supported in MOlliE. Then, we present the symbolic music encoding currently provided by the framework, and the available music collections used as reference for benchmarking. The chapter concludes with a large scale experiment of all the main architectural types of music models supported in the framework on a large dataset of symbolic music, thereby providing insights into (i) the influence and suitability of different modelling choices when looking at the statistical properties of a music collection; (ii) and the evaluation workflow we expect future modellers to follow thanks to MOlliE.

Chapter 6 describes our novel method for hierarchical music structure analysis – the musical structure communities (MSCOM) algorithm. First, the method is formally described in all its steps: from the extraction of audio features and the derivation of the music graph, to the multi-resolution community detection procedure meant to detect music structures at all possible levels. Then, MSCOM is compared to the current state of the art techniques for segmentation accuracy and its key properties are analysed from the resulting segmentations.

Chapter 7 extends the previous contribution and presents a novel method for measuring the structural complexity of music. It introduces a set of metrics encoding specific properties of the hierarchical segmentations produced by MSCOM. To test the effectiveness of these metrics, an experimental evaluation was carried out on a dataset comprising music with different degrees of structural complexity. Finally, the chapter shows and provides examples on how these metrics can be used for the automatic evaluation of structural complexity.

Chapter 8 investigates the use of structural properties of the music signal in the domain of music emotion recognition. The first section explores the use of the structural complexity metrics for valence and arousal regression (Chapter 7). Our goal is to test if any relationships can be found between structural complexity, as measured by our metrics, and music emotions. This is done via an experimental methodology where different regression methods are trained to predict valence and arousal solely from these metrics. The second and last section of the chapter investigates the use of another yet radically different structural property of music – the instrumentation of an audio recording, to improve the accuracy and the interpretability of MER systems. First, a novel neural network architecture leveraging music source separation techniques for isolating each voice and learning source-specific features for MER is presented. Then, the architecture is compared to the current state of the art MER techniques and a preliminary analysis of the model is carried out to demonstrate the increased interpretability.

Finally, Chapter 9 concludes with a summary of the contributions and a critical discussion on the obtained results. The primary research questions addressed in this thesis are recalled to evaluate the outcomes of our contributions achieved in the presented experimental studies. Conclusions, final remarks and a roadmap for future research directions close this thesis.

Part II

Background

Chapter 2

Prelude to computational music analysis

The field of computational music analysis, intersecting several areas of music information retrieval, includes a variety of tasks and applications handling music data in different formats – from beat, chord, and key detection, to music source separation and automatic music transcription. Nonetheless, this chapter focuses on three specific tasks that are particularly relevant to this thesis, namely: music modelling, music structure analysis, and music emotion recognition. The first task is at the core of the main research problem motivating this thesis – the challenge of learning long-term structure from music; the second one is a fundamental component of our methodology for the automatic evaluation of structural complexity; the latter is instead relevant for the investigation of structural features in music emotion recognition. Each section provides an overview of each task, and gives relevant literature for further treatment.

2.1 From sequence modelling to music modelling

The ability to model sequences is one of the major problems in unsupervised learning, and more precisely in predictive analysis, personalised drug design, material discovery and information retrieval. In this context, a properly designed model capable of capturing the underlying distribution to which the training sequential data is subject, is a powerful computational tool which enables several types of well-known tasks in machine learning, such as *event prediction* and *sequence generation*, as well as a deeper understanding of the phenomenon under study, e.g. discovery of relationships and patterns among features and events.

Although the nature of the sequential data differ from domain to domain, all of them can be cast under the task of sequence modelling, which is now presented from a general perspective. Given an i.i.d. dataset $\mathcal{D} = \{\mathbf{s} = (s_1, \ldots, s_N) \in \mathcal{A}^N\}$ containing sequences of tokens[1] $s_t \in \mathcal{A}$ defined over a vocabulary $\mathcal{A}$ and of arbitrary length N, we are interested in probabilistic models

[1] In this manuscript, the words *events*, *tokens*, *observations*, *sequence elements* are considered as synonyms.

over sequences such that the probability of each sequence $p(\mathbf{s})$ can be factorised as the product of token-wise probabilities as follows:

$$p(\mathbf{s}) = \prod_t p(s_t|\mathbf{s}_{<t}), \tag{2.1}$$

where $\mathbf{s}_{<t} = (s_1, \ldots, s_{t-1})$ represents the sub-sequence (or history) of the previous events. In the context of natural language processing, where events/tokens correspond to words (or characters), this task is commonly known as *language modelling*. Indeed, a word-level language model is defined as a conditional distribution on the identity of the t-th word w_t in a sequence, given a contextual representation of the identities of all the previous ones $\mathbf{w}_{<t}$.

As music is a form of language (Mehr et al., 2019), it deserves to be studied as a separate sequential phenomenon, hence the name of *music modelling*. To introduce and formalise the problem, let us focus on symbolic music, i.e. music stored in a notation-based format (e.g., MIDI, piano-roll, text) and containing explicit information about note onsets and pitch on individual tracks (but no actual sound). From a general perspective, assuming to fix a specific note duration (e.g. 16th) as the basic unit of a time step, a music model[2] should predict the next pitch (or the next group of pitches) to be played in the current time step, given all the previous ones. This is somehow analogous to a person asked to predict the immediate continuation of a piece after having listened an initial part of it. To illustrate the increased complexity of the computational task when compared to other sequence modelling problems, and emphasise the relation with Equation 2.1, we present the three main case studies of music modelling. These sub-tasks are generally known as: *monophonic melody modelling*, *polyphonic melody modelling* and *multi-track music modelling*[3]. The order in which these problems were presented intentionally captures an increasing level of complexity.

The easiest form of symbolic music modelling deals with monophonic music and it is commonly known as *monophonic melody modelling*. In this sub-problem, the objective is to model musical sequences of non-overlapping notes, i.e. at a give time step there is either a single pitch or a rest (Figure 2.1). This scenario perfectly aligns with Equation 2.1 and there are no particular differences with the task of language modelling.

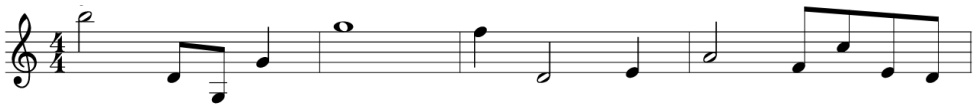


Figure 2.1: Excerpt of monophonic music (for *monophonic melody modelling*).

[2] In the literature, music models are also known as *predictive models for music*.

[3] The name of these tasks follows the general conventions in the computational literature, hence the terminology might be perceived as oversimplified or incorrect in the music domain. This also applies to several other music-related terms in this thesis, which are contextualised whenever any potential ambiguity might arise.

The second case study is more general, as it contemplates the possibility of multiple pitches being played at the same time step, though we are still modelling musical sequences associated to a single instrument/track. For instance, a musical part for piano typically follows a two-staff format, i.e. bass staff and treble staff, for left and right hand respectively (see Figure 2.2). In turn, according to the physical properties of an instrument, a single staff may contain chords made of two or more pitches (chords). Since we are still dealing with a single instrument, the same pitch cannot appear twice at the same time step, so the note overlap is only allowed for distinct pitches. From a probabilistic perspective, the polyphonic music scenario is more challenging to model, as it represents a generalisation of language modelling in the presence of higher dimensional sequential data. To make a comparison, we can try to visualise some text with overlapping words intentionally placed by the author to enrich their semantic and aesthetic load; although it would not be possible for a person to utter such intricate compositions, for music instead, a performer can achieve that by means of a polyphonic musical instrument.

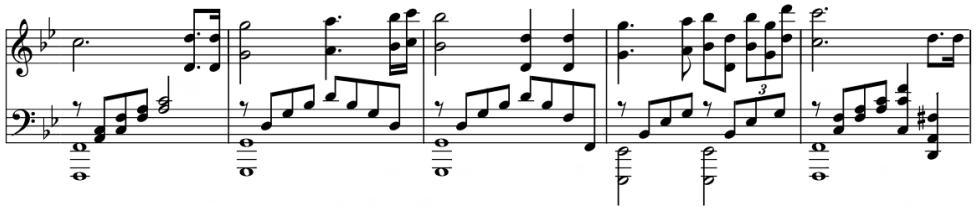


Figure 2.2: Excerpt of polyphonic music (*polyphonic melody modelling*).

Finally, the most general and challenging sub-task in music modelling involves dealing with symbolic music written for multiple instruments (or voices). As can be seen in Figure 2.3, each instrument may have up to two staves (in this example, one for the violin and two for the piano), hence the same pitch can occur more than once at the same time step. This last observation rules out the possibility to use an extended vocabulary unless multiple copies of the same pitch (one for each voice) are considered (Simon et al., 2018). Given the high complexity of the multi-track settings, only a minority of the current contributions seek to address this problem.

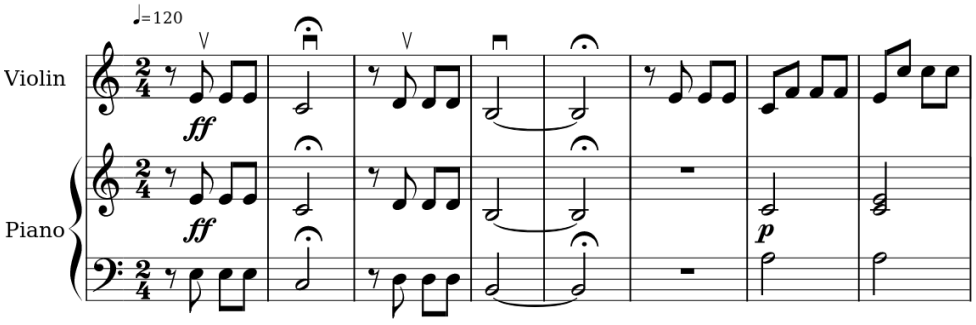


Figure 2.3: Excerpt of multi-track music for violin and piano (*multi-track music modelling*).

Despite other relevant tasks dealing with symbolic music exist, such as *melody accompaniment* and *style transfer*, this work focuses primarily on the aforementioned tasks as they constitute specific instances of the sequence modelling problem in the music domain. In addition, as argued in Section 2.1.2, music models can potentially support a wide range of symbolic music tasks due to their probabilistic nature, including those mentioned before.

2.1.1 Music modelling vs language modelling

Music and text are both two types of sequential processes, and we have already noticed how, from the modelling point of view, the two corresponding problems – music modelling and language modelling, exhibit different challenges due to the intrinsic properties of these phenomena. For instance, we argued that language modelling can be considered as a specific case of melody modelling (i.e. words as notes, sentences as monodic musical sequences), whereas the polyphonic and the multi-track scenarios introduce a level of generalisation that cannot even be found in text. The aim of this subsection is to support this last argument through the following observations, collectively contributing to the technical difficulties of the task.

- A note in a musical work contains various properties, such as *position*, *pitch*, *length*, and *intensity*. The overall emphasis of each property and their correlation can significantly vary depending on the type of music, which makes it difficult to model (Shin et al., 2017). Whereas we can easily find the counterparts of *pitch* and *position* in textual data, language modelling does not deal with other complex features such as *length* and *intensity* (e.g. a given word has fixed length and intensity at the text level, although the same cannot be said for audio-based language modelling and speech recognition).

- Music is not merely a series of notes, but entails a complex and hierarchical structure that is particularly challenging for any sequence model (Dieleman, 2015). Most contemporary music exhibits harmonic structure based on chord progressions, i.e. temporal sequences of chords. On a coarser timescale, note sequences can be grouped into bars, motifs, themes and phrases, sections, movements, and so forth. This nested organisation of musical ideas, according to their temporal granularity and articulation, is also more emphasised for certain genres. For example, classical music adheres quite rigorously to musical forms (e.g. the sonata form), providing a sort of layout of the large-scale temporal structures expected in a composition. Contemporary music instead, makes extensive use of the *verse-chorus-verse* structure – a specific type of sectional form.

- Conversely to language modelling, the hierarchical structure of music has further implications on the number and type of the resulting dependencies. Not only does music

develop through time, but a second dimension also originates when considering the superimposition of notes (polyphony) and/or instruments. Therefore, dependencies can be: (i) temporal/horizontal and with respect to a single voice (*intra-track dependencies*); (ii) vertical, throughout all the simultaneous pitches (*inter-track dependencies*); (iii) and diagonal – across time and harmony (*cross dependencies*). More formally, Briot, Hadjeres, and Pachet (2017) describe the former two dimensions as separate levels for musical content, which are defined as follows.

- *Sequence level* or *horizontal dimension* – at the level of a sequence of items (the actual temporal dependencies among successive notes or group of notes). In this case, the distribution is about the sequence of notes, i.e. the probability of a specific note occurring after the previous musical content in the sequence.
- *Item level* or *vertical dimension* – at the level of a compound musical item occurring during a single time step. In case of polyphonic music modelling, the distribution is about the relations between the components of a chord, describing the probability of notes occurring together (chord conditional distributions). In multi-track music modelling, the vertical dimension is further extended by the contribution of each musical part (e.g. a symphonic orchestra may have more than 90 instruments).

To complement Briot et al.'s perspective, we also include the *diagonal dimension* – spanning across both horizontal and vertical dimensions. As an example, let us consider the situation when an instrument plays a 2-bar motif which is then repeated by another instrument at a different interval (e.g. I-V interval) and before the end of the motif. In this scenario, not only the second instrument should be aware of what was played so far (from all the instruments), but a synchronisation with the first instrument is also required before it can start repeating the motif. The diagonal dimension is thus particularly emphasised during improvisation, as the performer should be aware of the current musical context and have the ability to anticipate/predict what the other performers are going to play next.

Considering the rich structural texture of music, even a simple pop piece may have more articulated dependencies than a novel. In sum, all these considerations suggest that music modelling is a rich and complex problem that is worth to be studied independently, especially due to its capacity to generalise other sequence modelling problems.

2.1.2 Why modelling music is important

As reiterated in Figure 2.4, both language and music modelling are two particular instances of the sequence modelling task. The difference, and the resulting complexity of these problems,

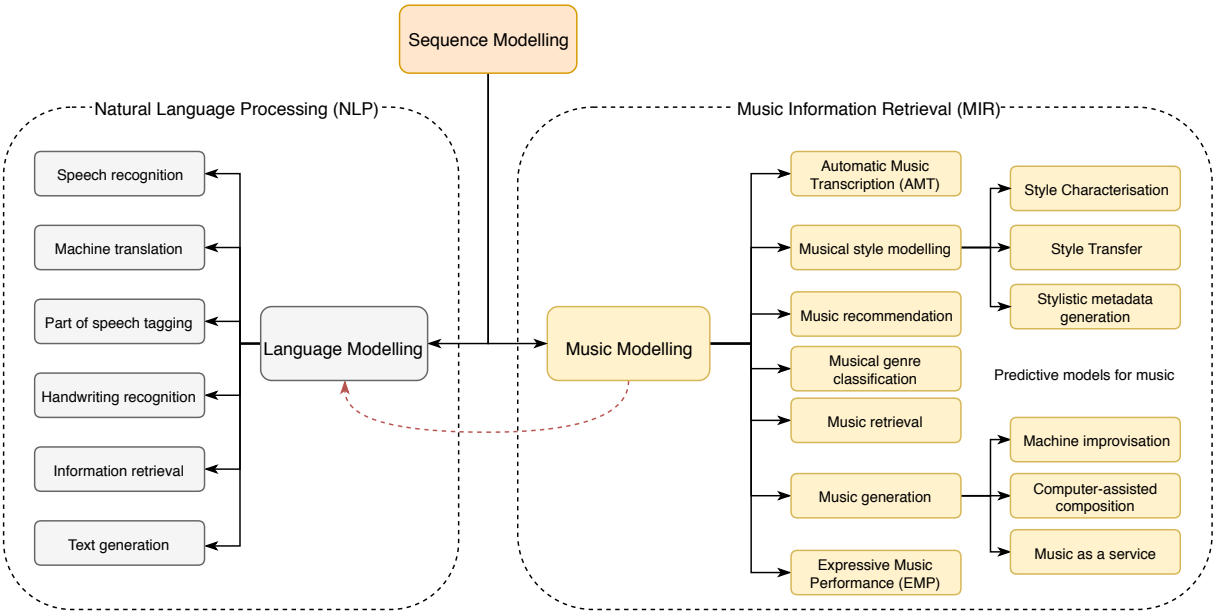


Figure 2.4: Examples of tasks related to language (left) and music modelling (right).

lies in the type of sequential data to be modelled as well as the possible application domains and sub-tasks that would benefit from their solutions. Notably, improving the performance of language model would bring tangible benefits to several natural language processing (NLP) tasks, such as speech recognition, machine translation, part-of-speech tagging, parsing, handwriting recognition, information retrieval and text generation. Each of these problems already identifies a research area per se. Nevertheless, the additional support of a language model can provide an intermediate layer to disambiguate among possible textual sequences/responses that are relevant for a task. This synergy among language models and task-specific methods was already demonstrated to further increase the performance of the resulting composite models, way before the advent of modern NLP architectures, e.g. for speech recognition (Graves, Mohamed, & Hinton, 2013). Nowadays, it is not a secret that BERT-like language models (Devlin, Chang, Lee, & Toutanova, 2018) are extensively used in virtually any NLP-related project.

Analogously, improving the predictive capabilities of music models – their ability to predict music, would benefit a wide range of problems and tasks in MIR. Examples of such tasks in the literature include, but are not limited to, are usually studied in the following research domains.

- *Automatic Music Transcription (AMT).* Whereas language models can be combined with acoustic models to improve speech recognition systems (Graves et al., 2013), symbolic music models can effectively support the automatic transcription of music from raw audio (e.g. live recordings). More precisely, AMT is defined as the problem of extracting a symbolic representation from music recordings – usually in the form of a time-pitch representation (e.g. piano-roll) or in a MIDI-like format (Ycart et al., 2017). Despite

the recent advancements in machine learning, AMT remains one of the most challenging problem in MIR, especially on polyphonic and multi-track music. Although there has been extensive research on acoustic models for music transcription, the complementary use of music models have received little attention until quite recently.

- *Musical style modelling*. In the symbolic domain, this task aims at learning a computational representation of the musical surface from notated music that captures representative composer- or genre-specific stylistic features. These stylistic features, characterising one or more musical pieces, are usually hidden from the surface – becoming apparent in the way that higher-level abstractions, such as patterns of rhythm, melody, harmony, and polyphonic relationships, appear in the music, interleaved and recombined in a redundant fashion (Dubnov, Assayag, Lartillot, & Bejerano, 2003).

  In this domain, music models can be trained in a way to automate the extraction of stylistic features from large symbolic corpora, rather than having to formally define rules that can explain them. In doing so, as similarly done in computer vision and multimedia content manipulation, the musical style of a certain composer can be "transferred-morphed" to any other musical pieces. Balancing the tradeoff between musical content and musical style, thus enables the development of novel creative applications in digital arts.

- *Music recommendation, retrieval and classification*. As the music industry has shifted towards digital distribution, all these tasks have increasingly gained attention from both the academic and the private sectors. As the name suggests, *automatic music recommendation* allows listeners to discover new music that matches their taste and preferences – also enabling online stores and streaming services to target their wares to the right audience Van den Oord, Dieleman, and Schrauwen (2013). In a similar vein, *music retrieval* refers to the problem of finding a collection of musical pieces matching some user-defined search criteria such as *explicit musical content*, to find music pieces or excerpts matching symbolic music queries (e.g. a motif); *mood* – retrieving music that suits a certain atmosphere; *emotion* – finding music that reflects or contradicts a given emotional state; and *genre and style* – to retrieve music that can be attributed as belonging to a particular genre and/or style (Veltkamp, Wiering, & Typke, 2008). At the core of these tasks lies the necessity to classify the musical content at large scale, using a set of semantic tags that accommodate retrieval/recommendation pipelines. In this context, music models are trained or fine-tuned on symbolic or audio tracks to predict a predefined list of classes (either genres, styles, moods, emotions, etc.) – a typical music classification task.

- *Automatic music music*. The problem of automatic music composition has always been a historical challenge for both composers and researchers, as it includes a wide range of

non-trivial tasks, such as the generation of melody, chords, rhythm, and lyrics – virtually any possible musical dimension that is pertinent for the desired musical context. One of the first computational attempts to artificial music composition dates back to 1959, with Hiller and Isaacson (1959) investigating the use of Markov chains as predictive models for music. Thirty years later, with the advent of artificial neural networks, Mozer (1994) introduced CONCERT – a system combining recurrent neural networks with a set of predefined rules from music theory to enforce the generation of music with more coherence and correctness. As detailed in Chapter 4, deep learning models have been extensively used for automatic music generation. To our best, although recurrent networks have been the predominant paradigm to date, the architectural types of automatic composition systems are rather diversified – ranging from restricted Boltzman machines and autoencoders, to generative adversarial neural networks, self-attention networks, and also including hybrid solutions based on deep reinforcement learning methods and evolutionary algorithms.

Exploiting the generative capabilities of music models not only opens up the possibility of investigating new types of music, but is also considered as a pinnacle to understand machine creativity (Brunner et al., 2017). In addition, several application domains stemming from automatic music composition, such as those listed below, are specifically addressed to composers, performers, and general users.

- *Computer-assisted composition*, describing a family of systems enabling artists and composers to augment their creative possibilities by suggesting possible "musical paths" as the continuation of a given piece. Examples include DeepBach (Hadjeres, Pachet, & Nielsen, 2016) and the FlowComposer (Papadopoulos et al., 2016).
- *Machine improvisation* defines a category of intelligent systems that can temporarily replace a performer during a live session for music continuation (Mann, 2016), or to simulate an ensemble of instruments playing together with a human performer in real time. In the latter case, an ensemble prediction model[4] predicts the actions of other virtual members of a musical group in order to provide proper accompaniment to a live soloist. Nevertheless, the use of these systems is not only restricted to live performances, since they can be potentially employed in the context of musical practice. In fact, individual musicians often engage in simulated ensemble experiences such as practice and performance with a fixed backing track and using looping effects (Martin et al., 2018). With predictive models, these experiences can be made reactive to the changing behaviour of the performer.
- *Music as a service* is another music generation use case with radically different

[4] In this context, the term *ensemble* refers to a group of instruments/musicians or voices.

objectives than the previous ones. Basically, the term describes all those systems that can generate full musical pieces according to some user-defined requirements. Technically, music as a service does not augment artists, but potentially replaces them – thus raising serious ethical concerns (B. L. Sturm, Iglesias, Ben-Tal, Miron, & Gómez, 2019). Considering the appealing economical potential of these systems, music as a service is a fertile ground for startups, such as Aiva[5], Jukedeck[6], and Amper[7]. At the time of writing, the main goal of these services is to provide multimedia content creators with a simple and inexpensive platform (e.g. a web app) where they can generate music to complement their digital works. For instance, Aiva can compose emotional soundtracks for films, video games, and commercials.

- *Expressive Music Performance (EMP)*. According to Widmer and Goebl (2004), a music performance can be defined as the intrinsically human act of interpreting, structuring, and physically performing a piece of music from sheet, memory or via improvisation. Such a complex human activity has several facets: physical, acoustic, physiological, psychological, social, and artistic. The purpose of computational models of EMP is to learn, from human performances, how a certain composition could be interpreted – by making explicit changes to the musical attributes of notes and introducing/altering rests (silence segments). In particular, an EMP system models the physical parameters defining a performance (e.g., onset timing, inter-onset intervals, loudness levels, note duration, etc.), to quantify (quasi-)systematic relationships between properties of the musical score, the performance context, and an actual performance of a given piece.

Besides the application scenarios mentioned so far, the task of music modelling already represents a well-known and challenging testbed to evaluate the memory capabilities of sequence models (Boulanger-Lewandowski et al., 2012). Therefore, advancements in music modelling would potentially provide new solutions that can adapted and reused for other sequence modelling problems, thereby representing a contribution to machine learning research.

2.1.3 Symbolic music formats and datasets

To conclude the section, this part briefly outlines the most common symbolic music formats and datasets used to train and evaluate music models.

[5] https://www.aiva.ai/
[6] https://www.bytedance.com/
[7] https://www.ampermusic.com/

Common formats of symbolic music

To the best of our knowledge, despite recent attempts at raw-audio modelling (Dhariwal et al., 2020; Dieleman et al., 2018), most music models have been trained on symbolic music. The latter can be broadly defined as a digital representation of the musical content in a composition/performance, encoded as a stream of symbolic events/descriptors but providing no sound. Symbolic music is usually the result of a manual annotation or transcription process, but it can also be automatically sampled from instruments using proper digital interfaces, or generated by an AMT system directly from an audio recording (as introduced before). The most common symbolic formats of music are: MIDI, piano-roll and ABC (textual notation), which are outlined below[8]. Other notorious standards exists, specifically for music scores, such as MusicXML (Good, 2001) and MEI (Roland, 2002), although they are rarely considered for music modelling, possibly due to the large amount of data needed to effectively train these models.

MIDI (Musical Instrument Digital Interface) is a technical standard "*that describes a protocol, a digital interface and connectors for interoperability between various electronic musical instruments, software and devices*" (Association et al., 2005). A MIDI file contains temporally-ordered events providing basic instructions to play a piece, but no actual sound. These events can be thought of as messages specifying note information (e.g. pitch and velocity), and control signals for adjusting parameters such as volume, vibrato and clock signals. Technically, the standards specifies five types of messages. The most common is represented by *Voice Channel* messages – used to transmit real-time performance data. Examples of these messages include *note-on* and *note-off* events, which are outlined below.

- A *note-on* message specifies information to play a note, including the channel number (an integer within [0, 15]), a MIDI note number (the note's pitch as an integer within [0, 127]), and a velocity value indicating how loud the note is played (another integer within [0, 127]). For example, playing a middle C (MIDI pitch 60) on the first channel (associated to a specific instrument) with velocity 30 would be encoded as:

$$< \text{Note on}, 0, 60, 30 >$$

- A *Note-off* message simply indicates the release of a note, with the velocity field denoting how fast this event should happen. For example, releasing the same middle C (on channel 1) with velocity 20 would analogously correspond to the following message:

$$< \text{Note off}, 0, 60, 20 >$$

[8] For a detailed overview we refer to (Briot et al., 2020)

Each note event is embedded into a track chunk – a data structure containing a delta-time value providing the timing information of the event. Timing is typically expressed as the number of ticks passed from the beginning of the sequence, where the granularity of such a temporal division is found in the MIDI header as the number ticks per quarter note.

Depending on the final application, MIDI files are generally processed and encoded in other forms to be easily consulted, understood and visualised. These transformations can be useful for analytical purposes, but also enable algorithmic procedures for metrically re-aligning a transcription by detecting and correcting timing mistakes. A common encoding of symbolic music that can accommodate these use cases is the *piano-roll*. This format is inspired from *automated-pianos*, which used a continuous roll of paper with punched perforations as input for a mechanically-automated piano performance. Similarly to a computer program, each perforation expresses a particular instruction – representing a note control information for a specific note (depending on its position on the roll), and with the length of the perforation denoting its actual duration. Hence, a piano roll is formally represented as a matrix where each row is associated to a pitch and columns correspond to successive windows of fixed duration, according to a predefined temporal discretisation of the piece (alias quantisation). Time can either be absolute (e.g. 10ms per frame/column) or relative (e.g. the 16th note).

To conclude, symbolic music can also be represented in textual format, as done in the ABC notation – a de facto standard for folk and traditional music. An ABC document typically includes a header of six lines to represent the metadata of a piece. Each of these line starts with a letter, identifying the type of information provided: T stands for the title of the piece, M for the meter or the time signature, L reports the length, whereas K indicates the key. As can be seen from the excerpt reported below, a textual representation of a melody then follows.

```
|:eA (3AAA g2 fg|eA (3AAA BGGf|eA (3AAA g2 fg|1afge d2 gf:|2afge d2 cd||
|:eaag efgf|eaag edBd|eaag efge|afge dgfg:|
```

In ABC, the pitch class of a note in the melody is encoded following the Helmholtz pitch notation. Moreover, shorthand conventions for the pitch letter are be used to express variations. For instance, *A* corresponds to *A4*, *a* to an *A* one octave up, and *a'* to two octaves up. To express the duration of a note, additional characters are appended to the pitch letter. Assuming that the default length in the header is 1/8 (i.e. the eight note), *a* would simply indicate a note duration of 1/8; instead, *a/2* or *a2* would denote a note duration of 1/16 and 1/4 respectively. Concerning metrical information, the pipe symbol is used to delimit measures. Besides its simplicity, it is worth to remark that the ABC notation can only represent monophonic melodies.

Main datasets of symbolic music

This last part provides an overview of some of the most common *symbolic music datasets* that have been used for both music modelling and generation. The overarching aim here is to give an idea of the kind of data symbolic music collections usually provide, hence it is not intended as a comprehensive review of datasets in the field. Listed below are the four datasets from (Boulanger-Lewandowski et al., 2012) – a staple in music modelling.

- *PianoMidi*, a collection of 124 classical music pieces for piano in MIDI format. The MIDI files were collected from a digital piano by means of a sequencer. The dataset also provides both the corresponding digital scores and the audio recordings.

- *Nottingham*, is another dataset of polyphonic music comprehending a collection of 1200 British and American folk tunes.

- *MuseData*, a collection of 783 classical music pieces in MIDI format focused on music composed in Europe between 1700 and 1825.

- *JSB Chorales* (Allan & Williams, 2005) is a polyphonic music dataset consisting of the entire corpus of 382 four-part harmonised chorales by Johann Sebastian Bach.

The former collections have been extensively utilised in music modelling, and they are still quite popular in the machine learning field for benchmarking novel sequence models (as detailed in Section X, sequence models are commonly tested on sequential data of different kinds, including text and music). Further symbolic datasets which are less common in the literature, but are particularly interesting for the additional data they provide, are listed as follows.

- *Lakh MIDI dataset* (LMD), a collection of approximately 176581 MIDI files, 45129 of which were matched and aligned to entries in the Million Song Dataset (Bertin-Mahieux, Ellis, Whitman, & Lamere, 2011). LMD is addressed to large-scale music information retrieval, both symbolic (using MIDI files alone) and audio content-based (using information extracted from the MIDI files as annotations for the matched audio files).

- *Mutopia project*, a collection of 2105 pieces of classical and contemporary music transcribed and maintained by a community of artists. For each piece, both sheet and MIDI files are made available, together with a number of modern editions and arrangements.

- *Weimar Jazz Database* (WjazzD), a collection of 456 jazz solo transcriptions released in MIDI and lead sheet formats. The database also provides a detailed analysis of the

piece: number of notes, bars, choruses, mean tempo, event density, metrical event density, sincopicity, chordal diatonic pitch class histogram, pitch class histogram, midi pitch histogram, semitone interval histogram, etc.

- *TheoryTabs* is a website maintaining a collection of contemporary pop songs accompanied with an educational-oriented interface reporting the harmonic analysis of the piece, displayed alongside the melody in a piano-roll format. In particular, for a given piece, chords can be visualised in both absolute terms (key-dependent), such as Cmaj, Bmin, or according to their relative function with respect to the key (Roman numerals).

- *MusicNet* is a collection of 330 freely-licensed classical music recordings, annotated with over one million labels providing the precise timing of: each note in every recording, the instrument that plays each note, and the note's position in the metrical structure of the composition. Labels are obtained from the musical scores after having been aligned to their corresponding recordings by dynamic time warping.

- *Symbolic Music MIDI Data* is another collection introduced by Walder (2016) and including 20K classical music pieces from `classicalarchive.org`. Due to copyright, these files are only made available in a MIDI-like format, following a data transformation operated by the author. Not only does this transformation allow for the release of the music material, but the author also used it as an encoding for music modelling.

- *Session.org* is a collection of folk tunes (short musical pieces) in ABC notation. As the name suggests, the dataset originates from an on-line platform that is used for sharing and discussing music played in "traditional music sessions" (often Celtic and Morris). The collection does not only include music transcriptions, but also discussions, jokes, accompaniment suggestions, and so forth. Contributions vary in detail, with some being quite elaborate, e.g., specifying ornamentation, grace notes, slurs and chords. Most transcriptions are monophonic, but some of them provide transcriptions for multiple voices.

- *The Essen Folksong Collection* is a dataset of monophonic melodies comprehending 8473 folk songs from different countries. The largest categories are associated to Germany (5365 songs) and China (2246 songs); whereas the remaining songs are scattered across Europe (848), the Americas (13), Asia (5), and Africa (1). The original format of these melodies follows a textual representation called Humdrum and denoted as "***kern*".

2.2 Music structure analysis

The general goal of music structure analysis (MSA) consists in decomposing or segmenting a given music representation into patterns or temporal units that correspond to musical parts and to group these segments into musically meaningful categories (Müller, 2015). From this definition, we can identify two related sub-tasks that considered together make it possible to segment music effectively: the first problem is known as *boundary detection* and it aims at detecting the time instants where a transition from a music segment to the next one occurs; the second sub-task is called *structural grouping*, and it consists in labelling each segment according to their similarity or musical function.

As detailed by Müller (2015), there are different criteria for segmenting and structuring music, though most of them can be categorised according to the focus they give to the following properties: *repetition*, which rephrases the segmentation problem in terms of the identification of recurring patterns; *homogeneity*, such as consistent timbre, the usage of certain harmonies or the presence of a specific instrument within each individual section; and *novelty*, where segment boundaries are expected to coincide with sudden changes in musical properties such as tempo, dynamics, or musical key. Not only can music be segmented according to different criteria, but also humans seem to combine different segmentation cues in an adaptive and subjective fashion when asked to recognise and derive structural information from music. As a consequence, the identification of segment boundaries is an ambiguous task even for music experts (McFee, Nieto, Farbood, & Bello, 2017), and it is thus common for two listeners to disagree on the form of a piece of music. Indeed, since structural analysis records a listener's creative interpretation as much as their perception, objectivity is arguably an impossible goal for annotations (J. B. L. Smith, Burgoyne, Fujinaga, De Roure, & Downie, 2011).

The listener-dependent and context-sensitive relevance of different segmentation criteria discussed before make MSA an extremely challenging task when approached with computer-based systems. In this area (i.e. automatic music segmentation), the challenge is to design algorithmic procedures for automatically analysing musical form and several approaches have been developed and compared (Nieto & Bello, 2015). Because of this, in recent years it has become a focus for music information retrieval (MIR) researchers, not only due to its complexity, but also because it enables large-scale MSA and several practical applications. From a broad perspective, as outlined by Müller, Chew, and Bello (2016), computational methods for structuring and decomposing digitised artefacts into semantically meaningful units are relevant not only for music content, but also for general multimedia content such as speech and video. Indeed, decomposing a complex object into smaller units is of practical importance for several content-specific applications, as it often constitutes the first step for simplifying subsequent processing and analysis tasks. The structural decomposition makes it possible to obtain compact object

descriptions that can be efficiently stored, queried and transmitted, and it opens up novel ways for users to find and access music information in large, unstructured and distributed multimedia collections. Towards this direction, Kurth et al. (2005) introduced *SyncPlayer*, a framework for accessing music-related content that combines different modalities like acoustic, graphical and textual representations obtained from running music- and text-based retrieval methods on every track in a music database. Another prime example of an application exploiting automatic MSA for content-based navigation is *SmartMusicKIOSK* (Goto, 2006), providing a music interface that enables users to visualise the structure of a song via a music map, which is designed to highlight the chorus and the most repetitive sections. This functionality is particularly useful for trial listening, when users need to quickly determine whether they like a specific selection of music (e.g. recommended by a system, or simply discovered), so that they will listen to the filtered selection from start to finish. In that case, which is quite common in popular music, users tend to jump to the most salient parts of a song if an audio thumbnail or summary of that track is not available. To facilitate this process, *SmartMusicKIOSK* augments within-song browsing so that users can skip sections of a song by interactively changing the playback position while viewing the music map (Goto, 2006). This tool, originally designed for record shops, is now part of a larger system called *Songle* (Goto, Yoshii, Fujihara, Mauch, & Nakano, 2011), a web service that estimates not only the music structure but also the melody line, the beat structure, and the chords of songs available on the web and visualises all of them in a synchronised way during playback. These tools are called active-music listening interfaces, as they allow users to enjoy music in more active ways than conventional playback. We refer to Goto and Dannenberg (2018) for a comprehensive overview of these interfaces.

This is also valid for music producers, when automatic procedures for MSA will be integrated into digital audio workstations to help them navigate around a particular track in an ongoing project. Besides improving the workflow of music production, MSA algorithms can also facilitate the creation of mash-ups and remixes, with the potential to be exploited by DJs during live sessions. A notable example implementing this creative use case is the *Unmixer* (J. Smith, Kawasaki, & Goto, 2019), a recently introduced web service that also utilises a source separation method to enable users to extract loops from one or more uploaded tracks, remix them, and create mash-ups of loops from different songs.

As for scientific applications, the ability to reduce the complexity of music can benefit different tasks in the field of MIR, such as beat tracking, audio thumbnailing and automatic music composition, other than providing a framework for testing theories of music perception. Indeed, if we can identify the structure of a piece of music, at least in terms of phrases and measures, we would facilitate the task of beat tracking considering that, where repetitions occur in music, the beats in the two repetitions should correspond (Dannenberg, 2005). In the context of music

generation, as there is a close relation between the cognitive tasks of music analysis and composition, an MSA engine could form the basis of an algorithmic composition system, especially considering the lack of long-term structure in music generated by automatic procedures.

Nonetheless, most of the current methods in computational MSA can only estimate large-scale structural patterns from music, typically corresponding to the sectional level of a composition. The algorithms following this approach can only produce a single-level segmentation of a piece of music, resulting in a sequence of non-overlapping segments that are expected to match the sections of the given track (flat segmentation). In contrast, hierarchical segmentation techniques for MSA take into account the hierarchical organisation of music and can detect structural segments related to musical form at different time scales. These methods produce multi-level segmentations in the form of hierarchies, where each level offers a segmentation of the given piece at a specific level of granularity. For instance, a divisive method for hierarchical MSA might identify the sectional patterns of a composition at the first level of the hierarchy; then, each section would be decomposed into smaller segments to reveal sub-structures. This process is repeated recursively until the piece is completely decomposed into its most granular components (e.g. measures, notes, or beat-aggregated audio features) or a particular stopping criterion is met during segmentation. In this way, each level in the hierarchy offers a segmentation which is the result of a refinement of the segmentation performed at the previous level.

2.2.1 From feature extraction to music segmentation

The first step in the pipeline of an automatic procedure for audio-based MSA consists in choosing and extracting acoustic features which are related to those humans observe when determining the musical form of a piece. As observed by Bruderer, McKinney, and Kohlrausch (2006), the instrumentation and the timbral properties of a sound source are of great importance for the human perception of musical structure, and the same can be said for the pitch content, upon which harmonic and melodic sequences are built (Paulus, Müller, & Klapuri, 2010). Hence, two different types of audio features are often considered: *mel-frequency ceptral coefficients* (MFCC), encoding the timbral properties of the signal; and *chroma features* or *pitch class profiles*, describing the distribution of the harmonic content of the spectrum into a fixed number of bins corresponding to pitches of a musical scale.

Harmonic features alone have turned out to be effective mid-level representations in the context of MSA (Bartsch & Wakefield, 2005; Gómez, 2006); most of the works in the literature are based on the extraction and the subsequent analysis of chroma-based features, without taking into account the timbral properties of a recording. However, by studying the relationship between different audio features and human annotations, J. B. Smith and Chew (2013) demonstrated that a listener's attention shifts among these features throughout a piece, and using a

single audio descriptor would lead to undetected structural boundaries.

Representing musical structure

As similarity is a key element for detecting music form, the audio features of a given audio recording $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$ are then compared with each other in a pairwise manner. Comparing feature vectors $\mathbf{x}_i$, $\mathbf{x}_j$ is done via a similarity function s (e.g. based on the Euclidean or cosine distance) which enables the computation of a square and symmetric matrix defined as $S(i,j) = s(\mathbf{x}_i, \mathbf{x}_j)$. This is called a *self-similarity matrix* (SSM) and was introduced in the music field by Foote (1999) to visualise the musical structure of a given track. If a musical segment of length $m+1$, starting and ending at times t_i and t_{i+m} respectively, is repeated at a later time t_j, then the sub-sequence of feature vectors $[\mathbf{x}_i, \ldots, \mathbf{x}_{i+m}]$ should be identical to $[\mathbf{x}_j, \ldots, \mathbf{x}_{j+m}]$, with the corresponding entries in the SSM being maximised with respect to the co-domain of the chosen similarity function. This is represented as line segments (or stripes) symmetrically on either side of the main diagonal in the SSM.

The SSM is the starting point of most of the automatic procedures for MSA, and several works attempt to enhance this musical structure representation in order to facilitate the subsequent extraction of structural patterns. Common enhancements of an SSM include those making it robust against musical variations such as tempo changes and transpositions (Müller & Kurth, 2006; J. B. Smith & Chew, 2013).

Structural analysis methods

The current methods in the MSA literature are often categorised according to the taxonomy proposed by Paulus et al. (2010), which is based on the segmentation principles of repetition, novelty and homogeneity. The approaches belonging to the first group are based on the intuition that structural patterns usually repeat throughout a piece. As repeated segments are visualised as stripes on the diagonal and off-diagonals in an SSM, these approaches mostly rely on the extraction of such visual patterns (Lu, Wang, & Zhang, 2004; Müller & Kurth, 2006). These methods seem to be more indicated for popular music and also for the identification of the most representative segments of a recording (Goto, 2006). Novelty-based approaches instead aim at detecting the boundaries between two contrasting musical ideas. As proposed by Foote (2000), this is achieved with a novelty function computed as the cross-correlation between the SSM and a short-term checkerboard kernel. Peaks of the so-defined novelty function indicate potential structural boundaries. Finally, homogeneity-based methods assume a strong inner acoustical integrity of segments with respect to the chosen musical features (Jensen, 2006). Homogeneous passages are associated to block-like regions in an SSM, and are usually detected by utilising

clustering algorithms. If a functional role is assigned to each homogeneous group, another approach is to define the structural segments as states of a hidden Markov model (HMM) (Levy & Sandler, 2008; Peeters, 2003).

In addition, fusion methods take the best of both worlds by combining different segmentation principles with a single framework. For instance, Paulus and Klapuri (2009) attempt to capture homogeneity and repetition properties by a single probabilistic fitness measure, whereas Kaiser and Peeters (2013) apply a late fusion strategy on the segmentations produced by a repetition- and a novelty- based approach.

2.2.2 Computational methods for hierarchical MSA

To the best of our knowledge, there are only two automatic methods that are capable to produce hierarchical segmentations, both of which are part of the music structure analysis framework (MSAF) (Nieto & Bello, 2015): the *Laplacian Structural Decomposition* (LSD) (McFee & Ellis, 2014a), for both boundary detection and structural grouping, and the *Ordinal Linear Discriminant Analysis* (OLDA) (McFee & Ellis, 2014b) that can only perform the former sub-task.

Laplacian Structural Decomposition (LSD)

LSD generates hierarchies of fixed depth, where each layer i consists of $i+1$ unique segment labels. For each layer, this method first partitions the recording into a set of discontinuous clusters (segment labels), and then estimates segment boundaries according to changes in cluster membership between successive time instants. From a technical perspective, a network representing both the timbral and the harmonic similarities of a track is constructed. Then, after the computation of the Laplacian matrix from the adjacency matrix representing that graph, spectral clustering is performed on this structure. This is achieved by isolating the first k eigenvectors of the Laplacian matrix and running the *k-means* clustering algorithm on the resulting data. This allows the detection of k uniquely labelled segments in a track, so that repeating the same procedure for successive values of k, e.g. $k = 1, 2, \ldots, 10$, as done in (McFee & Ellis, 2014a), would produce a hierarchical segmentation of the corresponding track. LSD has two main limitations, which emerge from this incremental approach:

- although a level can have an arbitrary number of segments, the actual number of unique segment labels is always fixed, and it corresponds to the value chosen for k at that level;
- the resulting segmentation is not truly hierarchical due to the independence between levels; each one is indeed obtained from a separate clustering step, thus, a segmentation at the k-th levels is not necessarily a specialisation of the one at $k-1$.

Ordinal Linear Discriminant Analysis (OLDA)

The OLDA method (McFee & Ellis, 2014b) performs agglomerative clustering of time instants into segments, resulting in a binary tree with time instants at the leaves, and the entire recording at the root. Each layer i has $i+1$ contiguous segments, and the tree is automatically pruned based on the statistics of segment lengths and the overall track duration. This results in a hierarchy of variable depth, typically between 15 and 30 levels, where each level can be seen as splitting one segment from the previous level into two. Because OLDA only estimates segment boundaries, it has to be coupled with a structural grouping algorithm in order to label the detected segments. According to McFee et al. (2017); Nieto and Bello (2015), the 2D-Fourier magnitude coefficients method (Nieto & Bello, 2014) is the preferred choice, since it yields state of the art results in automatic prediction of flat segment labels.

2.3 Music emotion recognition

The ability of music to express and induce emotions (Juslin & Sloboda, 2011; Meyer, 2008) and act as a powerful tool for mood regulation (Schäfer, Sedlmeier, Städtler, & Huron, 2013) are well-known and demonstrable. Indeed, research shows that music listening is a commonly used, efficacious, and adaptable device to achieve regulatory goals van Goethem and Sloboda (2011), including coping with negative experiences by alleviating negative moods and feelings Kuntsche, Mével, and Berson (2016).

Crucial to this process is selecting the music that can facilitate the listener to achieve a determined mood regulation target, which often is not an easy task. In order to support listeners in this process, emotion-aware music recommendation systems became popular as they offer the possibility to explore large music libraries using affective cues. Indeed, recommending music based on the emotion of the listener at home (Dornbush, Fisher, McKay, Prikhodko, & Segall, 2005) or background music personalised for the ones present in a restaurant would provide a more personal and enjoyable user experience (Jaimes, Sebe, & Gatica-Perez, 2006).

At the core of these systems is music emotion recognition (MER), an active field of research in music information retrieval (MIR) for the past twenty years. The automatic prediction of emotions from music is a challenging task due to the subjectivity of the annotations and the lack of considerable data for effectively training supervised models. Song and Simon (2015) also argued that MER methods tend to perform well for genres such as classical music and film soundtracks, but not yet for popular music (Saari, Eerola, & Lartillot, 2010). In addition, it is difficult to interpret emotional predictions in terms of musical content, especially for models based on deep neural networks. Although a few approaches exist for interpretable MER (Chowdhury, Vall, Haunschmid, & Widmer, 2019), the recognition accuracy of these methods

is compromised, with the resulting performance loss often referred to as "cost of explainability".

2.3.1 A primer on music emotion recognition

Prior to introducing content-based methods for MER, we provide the reader with the fundamentals concepts of the task and refer to (Barthet, Fazekas, & Sandler, 2012; Kim et al., 2010; X. Yang, Dong, & Li, 2017; Y.-H. Yang & Chen, 2012) for a detailed overview.

Induced vs perceived emotions Perceived emotions refers to the recognition of emotional meaning in music (Song & Simon, 2015). Induced (or felt) emotions refer to the feelings experienced by the listener whilst listening to music. Some research also suggests that responses for induced emotion are generally positive, and responses for perceived emotion are more consistent (Gabrielsson, 2001)

Annotation system The conceptualisation of emotion with its respective emotion taxonomy remains a longstanding issue in MER (Song & Simon, 2015). There exist numerous emotion models, from miscellaneous (Mcadams, Vines, Vieillard, Smith, & Reynolds, 2004) to domain specific (Zentner, Grandjean, & Scherer, 2008), categorical (Ekman, 1992) and dimensional (Posner, Russell, & Peterson, 2005), with the latter two being the prevailing ones. Whilst the categorical model focuses on all the emotions evolving from universal innate emotions like happiness, sadness, fear and anger (Ekman, 1992), the dimensional model typically comprises an affective two-dimensional *valence-arousal* space. Valence represents a pleasure-displeasure continuum, whilst arousal outlines the activation-deactivation continuum (Posner et al., 2005).

Time scale of predictions Predictions can either be static or dynamic. In the former case, the representative emotion of a song is given by one valence and arousal value (Kim et al., 2010). Emotion annotations can also be obtained over time (e.g. second-by-second valence-arousal labels), thus resulting in dynamic predictions (Schmidt, Turnbull, & Kim, 2010).

Audio features Musical compositions consist of a rich array of features such as harmony, tempo, loudness and timbre and these all have an effect on emotion. Previous work in MIR has fuelled around developing informative acoustic features (Kim et al., 2010). However, as illustrated in other works (MacDorman, 2007; Mion & De Poli, 2008; Schmidt & Kim, 2010) and to the best of our knowledge, there exists no dominant single feature for MER.

2.3.2 Methods for content-based MER

The field of MIR has followed a similar path to other machine learning ones. Prior to the deep learning era, most methods relied on manual audio feature extraction. Huq, Bello, and Rowe (2010) give an overview on how musical features were traditionally extracted and fed into different architectures such as support vector machines, k-nearest neighbours, random forests, deep belief networks and other regression models. These methods were tested for MER on Russell's well-established arousal and valence emotion model (Posner et al., 2005).

With the recent advancements in the field, these were succeeded with deep learning methods. Such techniques, like binarised and bi-directional long short-term memory recurrent neural networks (LSTM-RNN) and deep belief networks, have also been successfully employed for valence and arousal prediction (M. Xu et al., 2015). Other methods again used LSTM-RNNs for dynamic arousal and valence regression, on the acoustic and psychoacoustic features obtained from songs (Coutinho, Trigeorgis, Zafeiriou, & Schuller, 2015). Most of these works stemmed from entries in the MediaEval emotion challenge (Aljanaki, Yang, & Soleymani, 2017b). Multimodality has also been an interest for this research community, where Delbouys, Hennequin, Piccoli, Royo-Letelier, and Moussallam (2018) looked into the combined use of audio signal and the lyrics of a musical track for MER. Again, deep learning methods such as LSTM-RNNs are at the core of the architectures proposed.

Nevertheless, an important factor in machine learning has been to build interpretable models, to make them applicable to a wider array of applications. To the best of our knowledge, only a few works have attempted to build an interpretable model for MER. In (J. Zhang, Xianglin, Yang, & Nie, 2016), different model classes were built over the extracted and selected features. These vital features were filtered and wrapped, followed by shrinkage methods. In (Chowdhury et al., 2019), a deep network based on two-dimensional convolutions is trained to jointly predict "*mid-level perceptual features*", related to emotional qualities of music (Aljanaki & Soleymani, 2018), with emotion classes in a categorical annotation space.

Chapter 3

Fundamentals of Recurrent Neural Networks

To provide the background needed for the forthcoming literature review in music modelling (Chapter 4) and for understanding the contributions of this thesis, the chapter provides a compact yet comprehensive overview of artificial neural networks (ANNs), mainly focused on recurrent architectures for sequence modelling. In particular, starting from Section 3.1, the main architectural types of recurrent neural networks are presented, together with the main issues jeopardising the learning long-term dependencies – the underlying problem hindering the generation of music with a realistic level of structural complexity. The chapter proceeds with Section 3.2, overviewing the main modelling choices for the design of an ANN – from activation and loss functions to common regularisation techniques. To complement these concepts, Section 3.3 outlines the common steps for training ANN/RNNs and optimise their hyper-parameters, whereas Section 3.4 provides an overview of the most common evaluation methodologies for sequential models. Finally, the chapter concludes with an extensive overview of current RNN architectures, thereby highlighting how the ML field has tried to address the problem of learning long-term dependencies in sequential data from a more general perspective (c.f. Section 3.5).

3.1 Breaking the static nature of perceptrons

Recurrent Neural Networks (RNNs) are defined as a class of ANNs that is characterised by the presence of recurrent connections within the model, making it possible to process sequential information. The intuition behind RNNs is similar, at least in the intent, to that of Convolutional NNs (CNNs), and it is known as *parameter sharing*. Indeed, sharing parameters across different parts of a model disentangles the network from holding redundant representations of its input; this also allows to extend and apply the model to examples of different forms and generalise

across them (Goodfellow, Bengio, & Courville, 2016). In the context of RNNs, this feature is crucial when a specific piece of information can occur at multiple positions within the sequence. What we want to achieve is known as *time-shift invariance*, which is analogous to *positional invariance* for CNNs. For instance, if spotting a cat in a figure regardless of its position is what CNNs seek to achieve by means of parameter sharing, RNNs use the same principle (but in a different way) to associate the same semantic meaning to the following sentences: "Yesterday, I came back to Manchester", "I came back to Manchester, yesterday".

Whereas Time Delay NNs (TDNNs) implement the same idea of parameter sharing in CNNs, but to process sequential information, the approach followed by RNNs is significantly different and unique in its kind. In short, the recursive connections in the model make each member of the output a function of previous members of the output. Therefore, considering that each member of the output is computed with the same update rule applied to the previous ones, the parameters of the network are actually shared while processing the same sequence. We can better "visualise" this approach by introducing the concept of a computational graph.

Unfolding computational graphs

Computational graphs are mathematical structures providing an intuitive tool to formalise computation. The paradigm is even used by deep learning frameworks, like Tensorflow (Abadi et al., 2015) and PyTorch (Paszke et al., 2019), to define the architecture of ANNs, organise the computation across the network, and keep track of gradients. In a computational graph, a node is associated to a variable – a tensor of any order (e.g. matrix, vector, scalar). In this context, an operation is defined as a simple function of one or more variables (nodes) and it is assumed to return only a single output variable. Thus, if a variable z is computed by applying an operation to variables x and y, then we draw a directed edge from node x to node z and another directed edge from node y to z. When the operation is not clear from context, we should seek to obtain the name of the function as an annotation of the corresponding output node[1]. Hence, an edge represents a variable dependency and a function argument in case it comes with an annotation.

We can now describe the idea of unfolding a recurrent computation into a computational graph with a repetitive structure (e.g. chain of events), so as to see that unfolding the graph corresponds to sharing parameters across the network. For notational purposes, we assume that our RNN operates on a sequence of vectors $x^{(t)}$ with t (the time step index) ranging from 1 to τ. However, it is worth mentioning that the time step index t does not necessarily imply any passage of time, even though it typically refers to the position of an event/token in the sequence.

If we restrict the focus to the way a sequence is processed and the information flows in the model, a recurrent neural network can be defined in the form of a *dynamical system*. Assuming

[1] As output node here we refer to the output of the operation and not to the output nodes of a neural network.

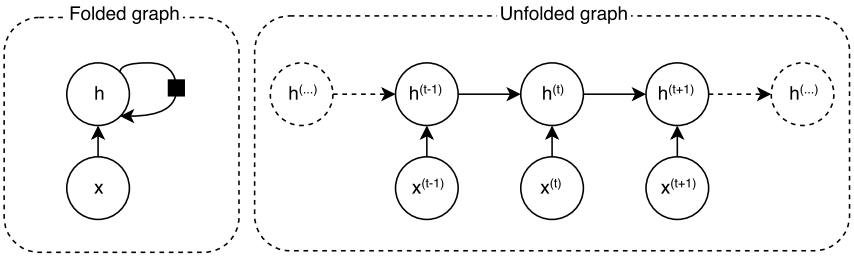


Figure 3.1: From the folded to the unfolded computational graph.

that we are dealing with recurrent connections in the hidden layer of the network,

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta) \tag{3.1}$$

represents the actual hidden state h of the network at time t, where θ denotes a parameter set used to compute the function f and typically corresponds to a tensor of weights. For a number of time steps $t \leq \tau$, the graph can be unfolded by applying the above definition $t-1$ times. Trivially, when $t = \tau$ the whole sequence is unfolded, thus recurrence disappears.

In accordance with the above definition, a recurrent network with no output units can be represented in two ways: as a folded graph (or circuit diagram) with a black square to indicate that an interaction takes place every single time step from $h^{(t)}$ to $h^{(t+1)}$; as an unfolded computational graph, in which a component is represented by many different variables, with one variable per time step, representing the state of the component at that point in time (Goodfellow et al., 2016). Both these representations are schematically reported in Figure 3.1.

Unfolding a recurrent network thus corresponds to the operation of mapping the circuit diagram resulting from the definition of that network to a computational graph with a size dependent to the length of the sequence we are dealing with. This operation allows us to define the state of the hidden unit after t steps with a function $g^{(t)}$ as follows:

$$h^{(t)} = g^{(t)}(x^{(t)}, x^{(t-1)}, \ldots, x^{(1)}) = f(h^{(t-1)}, x^{(t)}; \theta). \tag{3.2}$$

Therefore, the current hidden state results from the application of a function $g^{(t)}$ defined on all the elements of the past sequence, and should thus be able to summarise that sequence for the sake of the task we want to perform with the network. Moreover, since the unfolding process allows us to factorise the function $g^{(t)}$ into repeated applications of a function f, two main advantages over traditional MLPs are given: (i) the same network can handle sequences of arbitrary length; (ii) we can use the same transition (activation) function with the same parameters at every time step. Using the same model to process sequences of various lengths is generally associated with increased generalisation capabilities and less training data.

3.1.1 The 4 basic RNN architectures

Unrolling the computational graph of RNNs makes it possible to appreciate the benefits of their scalable architecture, as it allows us to visualise how the same reference model can be easily adapted to perform different sequence learning tasks. In fact, whereas traditional feed-forward ANNs can only process fixed-sized vectors as inputs and return fixed-sized vectors as output, RNNs are completely disentangled from this limitation.

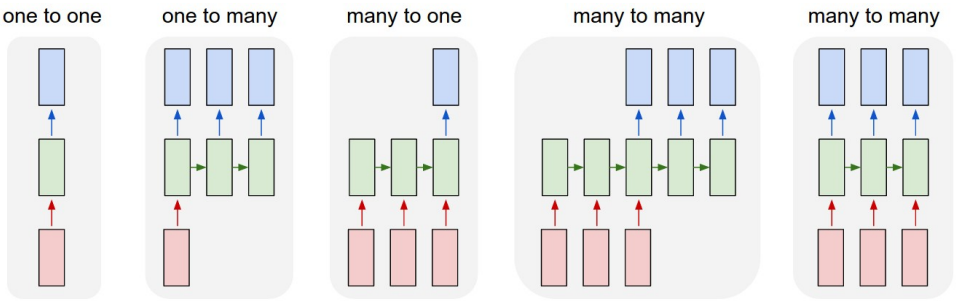


Figure 3.2: Traditional feed-forward ANN vs the 4 basic RNN architectures (from (Karpathy, n.d.)). Red, green and blue nodes denote input, hidden and output vectors, respectively.

The first and clearest taxonomy of RNN architectures is reported in Figure 3.2, and it is based on the most common sequence learning tasks. Nodes belonging to the unfolded computational graphs in this figure are coloured according to their role in the neural network.

As can be seen from the recurrent connections in the hidden layers, the state of the network at each time step is "enclosed" in the hidden layer; this consideration holds for every recurrent architecture presented in this section. Therefore, depending on the nature of the inputs, the expected output and how information flows within the network, we can identify the following 4 basic RNN architectures: (i) *continuous sequence to sequence learning* (e.g. video classification); (ii) *delayed sequence to sequence learning* (e.g. machine translation); (iii) *vector to sequence learning* (e.g. image captioning); (iv) *sequence to vector learning* (e.g. sentiment analysis). Conversely to traditional feed-forward ANNs, each of these RNN architecture does not impose any constraint on the size of input sequences (a vector can indeed be considered as a sequence of length 1) because the network can be unfolded as many times as need be.

Continuous sequence to sequence learning

This basic architecture is probably the most well-known. As illustrated in Figure 3.3, recurrence applies to the hidden units and the network produces an output at each time step. In other words, this RNN takes elements of a sequence to return a single output for each of them simultaneously. There is no delayed computation of the output, and this feature permits to achieve a kind of synchronisation between the input and the output sequence. As a result, this basic architecture

is generally suitable for all those tasks in which we need to provide an output as soon as the network is fed with a new observation[2]. An application domain of RNNs which adopts this basic architecture is video classification, where each video frame is labelled with respect to some predefined classes (e.g. anomaly states, facial emotions).

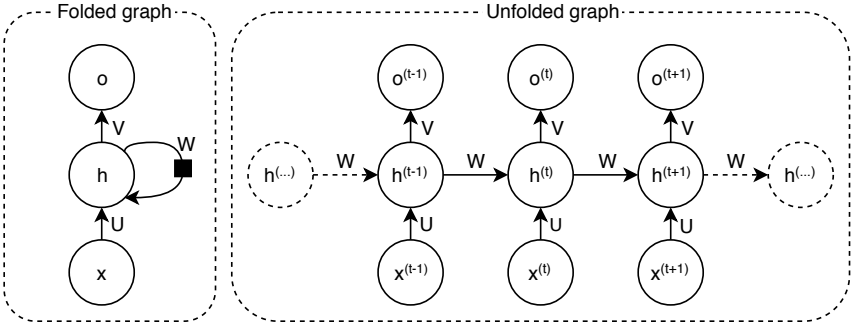


Figure 3.3: Continuous sequence to sequence learning architecture

The computational graph reported in Figure 3.3 generalises any sequence to sequence architecture and is useful to understand how the computation flows (forwards and backwards) within the network. More precisely, the basic architecture has 3 layers: input, hidden and output, with recurrent connections in the hidden layer. Hidden connections are weighted by a matrix U, the recurrent hidden-to-hidden connections are parameterised by a weight matrix W, and hidden-to-output connections are weighted by a matrix V. At each time step t from 1 to τ (the length of the sequence) forward propagation is achieved through the following update equations:

$$a^{(t)} = Wh^{(t-1)} + Ux^{(t)} + b \qquad (3.3) \qquad\qquad h^{(t)} = tanh(a^{(t)}) \qquad (3.4)$$

$$o^{(t)} = Vh^{(t)} + c \qquad (3.5)$$

where b, c denote the bias vectors of the hidden and the output layers respectively, and we are assuming the hyperbolic tangent ($tanh$) as an activation function for the hidden units.

In order to map an input sequence x to a task-dependent output sequence o, a loss L measures how far each o is from the corresponding training target y. This is done for every element in the sequence, i.e. at each time step t. To that aim, if using *softmax* outputs and assuming that o is the unnormalised log probabilities, the loss L internally computes:

$$\hat{y}^{(t)} = \text{softmax}(o^{(t)}) \qquad (3.6)$$

and compares this to the corresponding expected output, i.e. the target $y^{(t)}$. Therefore, the total loss for a given input sequence x paired with an output sequence y is defined as the accumulation of the losses over all the time steps. Formally, this is formulated as follows:

[2]Here, the terms *observation*, *event*, *token*, *element* of a sequence are often used as synonyms

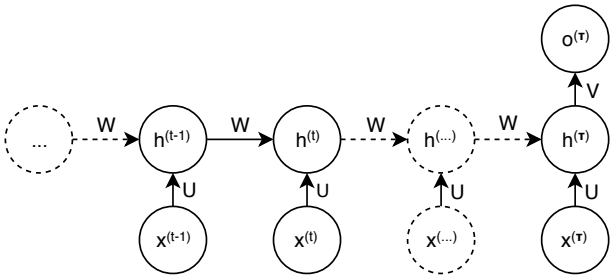


Figure 3.4: Sequence to vector learning architecture (Encoder)

$$L\left(\{x^{(1)},\ldots,x^{(\tau)}\},\{y^{(1)},\ldots,y^{(\tau)}\}\right)=\sum_{t}L^{(t)}. \tag{3.7}$$

As better explained in Section 3.2.2, the choice of loss function is strictly dependent on the task. In principle, any differentiable function that can effectively measure the quality of the network output with respect to either the targets (if available) or a specific task-dependent criterion, is potentially suitable.

To conclude, it is also worth mentioning that the class of sequence to sequence learning RNNs was shown to be Turing-Complete (Siegelmann & Sontag, 1995). Therefore, a sequence to sequence RNN of a finite size *can approximate* arbitrary programs with proper weights.

Sequence to vector learning

Although the sequence to sequence framework is the most general for RNNs, slightly changing/adapting it can lead to similar architectures that can accommodate other sequence learning tasks. In this case, we are interested in summarising the content of a sequence with a fixed-size representation, which is generally of interest for further processing. This is the intent of vector to sequence learning and the corresponding architecture is reported in Figure 3.4. Also known as encoders, these RNNs are restricted to outputting *one and only one vector* once the sequence has been fully fed into the network. In other words, a sequence to vector RNN needs to be completely unfolded before it can generate a single output – alias the *context* of that sequence.

The single output of an Encoder RNN for a given sequence can thus be viewed as a kind of summary or compressed representation of all the elements of that sequence. Ideally, this vector should be as rich as possible to capture the context of the sequence in full. An example of vector to sequence learning task is *sentiment analysis* where a given sentence is classified as expressing positive or negative sentiment (Karpathy, n.d.).

Vector to sequence learning

Besides their diversity for sequence learning tasks, the basic RNN models introduced so far are restricted to sequences as input. However, how could we adapt the general RNN architecture such that it can process a single vector as input? In the end, a single input vector can be considered as a sequence of length one. According to Goodfellow et al. (2016), we have three different modelling approaches for dealing with this: (i) considering the same input vector as an extra input at each time step (propagating the same input multiple times); (ii) considering the given input vector as the initial hidden state $h^{(0)}$; (iii) combining both approaches.

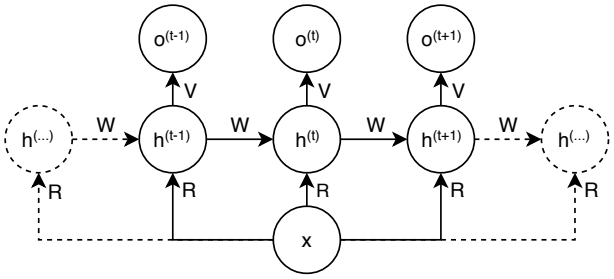


Figure 3.5: Vector to sequence learning architecture (Decoder).

The common approach is to introduce a new weight matrix R to parameterise the interaction (or combination) among the single input x and each hidden unit vector $h^{(t)}$. The same product $x^{\intercal}R$ is added as an additional input to the hidden units at every time step (Goodfellow et al., 2016). The resulting architecture thus allows us to map a fixed-length vector x into a distribution over sequences $\mathbf{Y}$. However, the architecture reported in Figure 3.5 makes a step further, as it assumes that the target sequence is available at training time. In this way, each target observation $y^{(t)}$ serves both as input at time t and, during training, as target (for the previous time step).

If we consider the input vector x as a sort of representation of a specific context, we are asking our vector to sequence RNN to derive an extended interpretation of that context. This operation is thus complementary to the learning task seen before. In fact, whereas an encoder tries to infer the context from a sequence, a vector to sequence RNN attempts to expand the compact representation of a given context in order to derive a sequence from it. These RNNs thus act as decoders, and an example of a vector to sequence task is image captioning (from a single image to a sequence of words describing the image).

Delayed sequence to sequence learning

The most evident limitation of the sequence to sequence recurrent architecture presented in Section 3.1.1 is that the input and output sequences always have the same size. As argued by Goodfellow et al. (2016), this problem comes up in several sequence to sequence learning tasks,

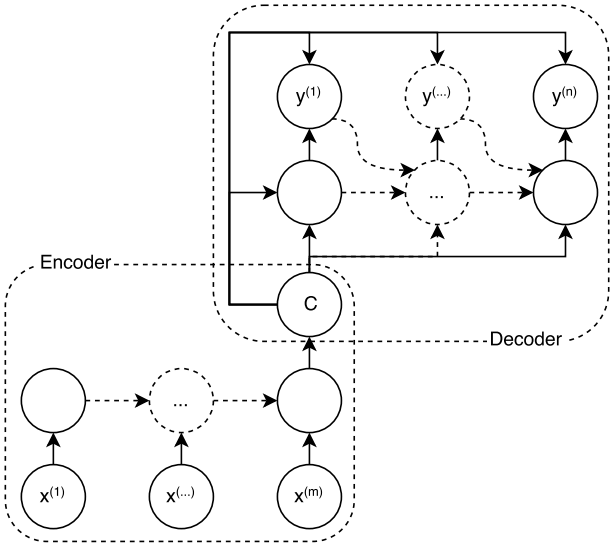


Figure 3.6: Delayed sequence to sequence learning architecture (Encoder-Decoder).

such as speech recognition and machine translation, where the input and output sequences in the training set are not of the same length. Even though we can find some tricks to deal with this problem, like introducing a special symbol denoting the absence of an output value, these approaches often lack generalisability and cannot be considered valid solutions.

To address this problem, Cho, Van Merriënboer, Gulcehre, et al. (2014) and Sutskever, Vinyals, and Le (2014) introduced a novel architecture – the Encoder-Decoder RNN. As the name suggests, the idea consists in combining a sequence to vector RNN (an encoder) with a vector to sequence RNN (a decoder), as the ones introduced before. More precisely:

1. an encoder processes the input sequence in order to derive its context C, i.e. a representation that effectively summarises the content of the input sequence $\{x^{(1)}, \ldots, x^{(m)}\}$;

2. a decoder takes the computed context C to generate the output sequence (or computes the probability of a given output sequence) $\{y^{(1)}, \ldots, y^{(n)}\}$.

As illustrated in Figure 3.6, the actual innovation of such recurrent networks is that the lengths m and n of the input and the output sequences can differ. To better perform on such sequence learning tasks, the encoder and the decoder are trained together to maximise the average of $\log P(y^{(1)}, \ldots, y^{(n)} | x^{(1)}, \ldots, x^{(m)})$ over all the input-output pairs x, y in the training set.

Nevertheless, the original architectures presented in (Cho, Van Merriënboer, Gulcehre, et al., 2014; Sutskever et al., 2014) still had a major limitation. In fact, the context C produced as output by the encoder has a fixed dimension, thus it might struggle to fully capture and summarise long sequences. To address such a limitation, Bahdanau, Cho, and Bengio (2014)

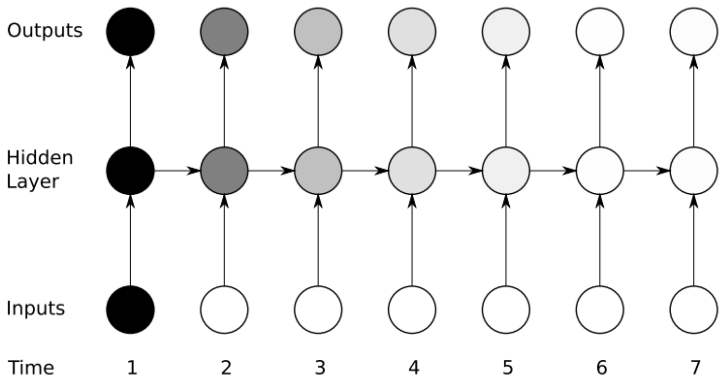


Figure 3.7: Illustration of the vanishing gradient problem: the contribution of the hidden unit at time 1 becomes weaker and weaker as the recurrent network is unfolded and completely vanishes after 5 time steps. Besides the simplicity of this example, this tends to happen normally after 5 to 10 steps of backpropagation in a vanilla RNN.

introduced a revised implementation of the Encoder-Decoder architecture in which the context C is a variable-length sequence rather than a fixed-size vector.

3.1.2 The vanishing and exploding gradient problems

Before resuming our journey on recurrent neural networks, to conclude this section we briefly outline the two most known and discussed problems in the literature of RNNs.

Recurrent neural networks are generally trained using one of the variants of gradient descent (i.e. batch, mini-batch or stochastic gradient descent) with a generalisation of the back-propagation algorithm called *back-propagation through time* (BPTT). In a few words, the high-level idea of this procedure consists in the application of the generalised back-propagation algorithm along the unrolled computational graph of a recurrent network. Thereafter, the so-obtained gradients may be used with any gradient-based technique to train the network, like those mentioned before; no further measure is required.

However, as a prominent paper explicitly mentions in its title (Bengio et al., 1994), training a recurrent neural network can be potentially inefficient for certain sequences and tasks because "*learning long-term dependencies with gradient descent is difficult*". In particular, as sequences are generally long in most of the problems worth studying, the resulting unfolded computational graphs of the possible recurrent networks are long to traverse during back-propagation. As a consequence, gradients propagated over many stages within the unfolded network tend to either vanish or explode. Although Goodfellow et al. (2016) argue that exploding gradients jeopardise the optimisation of the network parameters but are often rare to experience, the vanishing gradient problem is always "on the agenda" of RNNs. In fact, even if we consider a

recurrent network whose parameters are such that gradients do not explode when propagated, the difficulty with long-term dependencies is mainly due to the exponentially smaller weights given to long-term interactions compared to short-term ones.

More naively, we can imagine this problem as trying to hear what a far away person is saying, while another person closer is shouting. Therefore, when the gradient contributions from farther steps become zero, the state at those steps do not contribute to what we are learning (long-range dependencies are completely ignored by the network).

More formally, the vanishing and exploding gradient problems are both attributable to the fact that recurrent networks perform the composition of the same function, with the same weight matrix, once for each time step. Besides the fact that these compositions can result in extremely nonlinear behaviour, which is generally not desirable for parameter learning, the problem is still observable if we get rid of the non-linear activation function.

$$h^{(t)} = W^{\top} h^{(t-1)} = \left(W^{t}\right)^{\top} h^{(0)} \tag{3.8}$$

For instance, and for the sake of our investigation in these problems, the above equation describes a simple recurrent neural network with identity activation function and no inputs. Thus, assuming that the weight matrix W admits an eigendecomposition of the form $W = Q\Lambda Q^{\top}$ where Q is orthogonal, we can rewrite the definition of the hidden layer as follows:

$$h^{(t)} = Q^{\top} \Lambda^{t} Q h^{(0)} \tag{3.9}$$

Therefore, as can be noticed from the above equation, since the eigenvalues of the weight matrix decomposition are raised to the power of t, the two following scenarios are possible:

- eigenvalues with magnitude less than one will decay to zero;
- eigenvalues with magnitude greater than one will explode.

Hence, the vanishing and exploding gradient problems arise from the fact that gradients travelling back through the unfolded graph are scaled according to Λ^{t}. In practice, vanishing gradients make it difficult to know which direction the parameters should move to improve the cost function, while exploding gradients can make learning unstable (Goodfellow et al., 2016).

The vanishing gradient problem is not only found in RNNs, but it generally occurs in deep feed-forward ANNs. Nonetheless, this problem turns out to be definitely more common in recurrent networks because their computation tends to become deeper (in time) than other networks. In fact, recurrent networks are "deep" by definition, even without multiple recurrent layers stacked together: it is the length of the sequences which make RNNs inherently deep.

Addressing the vanishing gradient problem

As exploding gradients can be tackled more easily with regularisation methods, a great deal of effort has been accumulated by the research community to find proper and effective ways to address the vanishing gradient problem and learn long-term dependencies.

One possible way to learn both short-term and long-term dependencies simultaneously may be that of defining a recurrent architecture that is capable to operate at multiple time scales. In other words, what we seek to achieve with this approach is to let some parts of our model operate at fine-grained time scales (so as to focus on smaller details and capture short-term dependencies), while others can operate at coarser time scales thereby transferring information from the distant past to the present (i.e. capturing long-term dependencies).

The more traditional ways to realise such a "two-resolution model" can be classified into two main categories, according to the nature of their approach. In particular, on one hand we can find those approaches attempting to introduce or re-define the recurrent connections of neural networks (e.g. adding skip connections and removing connections); on the other hand, we can think of improving our recurrent units in such a way to hold more information from the past (e.g. leaky units). These basic approaches are outlined as follows.

- **Adding skip connections** is a simple architectural strategy for capturing long-term dependencies which consists in adding direct connections from variables in the distant past to variables in the present (Goodfellow et al., 2016). Using skip connections in a recurrent network thus corresponds to the idea of incorporating delays in our model. More precisely, whereas the recurrent connection in a traditional RNN connects a neuron at time $t-1$ with a neuron at time t (i.e. with unitary delay), a skip connection connects a neuron at time $t-d$ with a neuron at time t (i.e. with delay equal to d).

  As a result, since gradients in a traditional RNN may vanish or explode exponentially w.r.t. the number of time steps τ, gradients now diminish exponentially as a function of $\frac{\tau}{d}$. However, since our model contains both delayed and single step connections, gradients may still explode exponentially in τ. Therefore, event though not all long-term dependencies may be represented well in this way, empirical studies indicate that adding skip connections allows the learning algorithm to capture longer dependencies.

- **Leaky units** attempt to mitigate the vanishing gradient problem by re-defining the basic recurrent unit of traditional RNNs in such a way as to force them to retain more information from the past time steps. They are considered the precursors of today's more advanced recurrent units (e.g. LSTM and GRU) and are still relevant to date.

  Intuitively, a leaky unit can be defined as a unit with a linear self-connection and a weight

vector near one of these connections. More precisely, the activations of a hidden layer of linear units can be defined as follows:

$$h^{(t)} = tanh(\ \alpha(Wh^{(t-1)}) + (1-\alpha)(Ux^{(t)})\ ) \tag{3.10}$$

where $0 \leq \alpha \leq 1$ control the importance of the previous activations and it is known as leaky factor. Therefore, we are accumulating a running average of the hidden states: when α is near one, the running average remembers information about the past for a long time, and when α is near zero, information about the past is rapidly discarded.

Leaky units thus allow to capture long-term dependencies as similarly done by adding skip connections through time. However, whereas the latter requires the specification of a fixed time delay d, with leaky units we can achieve the same effect in a more smooth and flexible way, i.e. by adjusting the real-valued α. As for the selection of the leaky factor, we generally have two distinct options: treating α as a hyper-parameter (i.e. a value that is kept fixed during training); or promoting α as an actual parameter to learn.

- **Removing connections** is another approach to learn long-term dependencies based on the idea of letting information flow easily through long distances. We can think of this process as creating a "fast track" for information coming from more distant units.

  In particular, whereas skip connections add new (but delayed) edges, and units receiving such new connections may learn to operate on a long time scale but may also choose to focus on their other short-term connections, removing connections literally drops the basic 1-delayed connections and replaces them with longer connections. As a consequence, the resulting units are forced to operate on a long time scale.

  The idea of removing connections can be done in several ways and it does not follow a single procedure. In fact, we just need to ensure that a group of recurrent units is forced to operate at different time scales. For instance, we can consider to:

  1. make our recurrent units leaky, while ensuring to have different groups of units associated with different fixed time scales;
  2. perform explicit and discrete updates at different time steps, with a different frequency for different groups of units.

Overall, all these architectural approaches can be considered as the basic techniques available to mitigate the problem of learning long-term dependencies.

3.2 Elements of model definition

Defining am RNN is not limited to the choice of its baseline architecture. Similarly to other artificial neural networks, all the basic building blocks of the network should be carefully designed and composed such as: (i) the type of neuronal units in both the hidden and output layers; (ii) the loss function the model will attempt to minimise; (iii) the regularisation methods to prevent overfitting and improve generalisation; (iv) the training procedure for optimising the loss function; (v) the hyper-parameters of the model and so forth. This section focuses on those aspects that are more closely related to the architecture definition, namely (i) to (iii), whereas the remaining steps are outlined in the upcoming sections.

3.2.1 Types of neuronal units

As previously discussed, neurons in the hidden layer of an RNN compute the weighted sums of their current input $x^{(t)}$ and of their previous state $h^{(t-1)}$, in addition to a bias b to yield $a^{(t)}$ (Equation 3.11). Next, an element-wise activation function ϕ is applied to $a^{(t)}$ to control the extent to which neurons should fire (Equation 3.12); the so-obtained value $h^{(t)}$ is then broadcast to the outer connections for further processing. In simple terms, an activation function defines the "degree of excitement" of a hidden neuron w.r.t. the input received and its previous state.

Analogously, neurons in the output layer of a recurrent network perform the same operation but only on their input and bias, due to the absence of recurrent connections:

$$a^{(t)} = Wh^{(t-1)} + Ux^{(t)} + b \tag{3.11}$$

$$h^{(t)} = \phi(a^{(t)}) \tag{3.12}$$

where, as introduced in Section 3.1.1, the matrix U denotes the weights of the input-to-hidden connections, and W weights the hidden-to-hidden connections.

As the literature comprehends several activation functions, we intentionally generalised the function term ϕ in Equation 3.12. In practice, as neurons typically perform affine transformations from their incoming layer, the name of their activation function is typically used to identify their type. For instance, using a sigmoid activation, it is common to refer to a *sigmoid layer* or to *sigmoid units*. Generally, the most common activation functions can be grouped into the following two families: *linear* and *non-linear* functions.

Linear functions

The class of linear functions includes all those polynomials of degree equal to one, thus it represents the easiest but less effective solution to employ in a neural network. Formally, these functions are defined for a vector c which is multiplied element-wise to $a^{(t)}$ as follows:

$$h^{(t)} = c \odot a^{(t)} \tag{3.13}$$

These functions are just straight line functions where activation is proportional to their input. An example from this family is the identity function, which trivially leaves $a^{(t)}$ unaltered.

Although linear functions are easy to compute and differentiate, their complexity is much limited when compared to other activations functions, and this translates into their inability to learn complex functional mappings from data. Their limitations are summarised below.

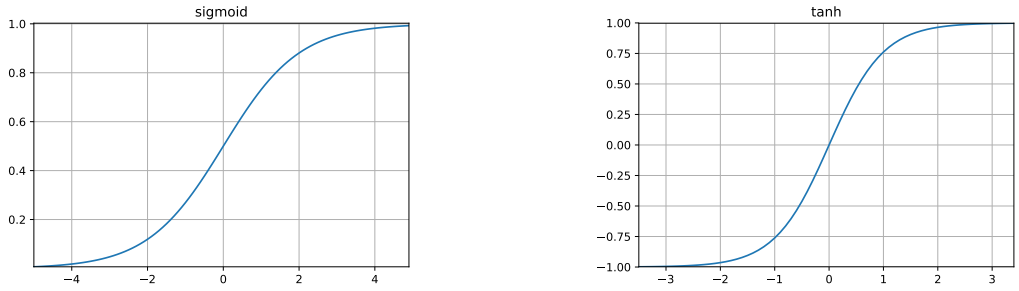
- A network with linear activations is hard to train, not just computationally, but for effectively learning parameters. More precisely, the derivative of these functions is always constant and the gradient has no relationship with $a^{(t)}$. As a consequence, the changes made during backpropagation are constant and not depending on the change of input.

- The second consideration is more profound and critical, because it contradicts the approach and the core intuition of deep neural networks. In particular, assuming that every layer in our network uses linear activations, the activation function of the last layer can be expressed as a linear function of the input of the first layer (a linear combination of linear functions is still another linear function). In other words, more linear layers can just be replaced by a single layer. Therefore, there is no advantage from stacking multiple layers: the whole network is still equivalent to a single layer with linear activations.

Such notorious drawbacks thus restrict the class of linear activations to simple problems, i.e. those in which data is already linearly separable. In fact, a neural network with linear activations would merely implement a linear regression model.

Non-linear functions

As the intuition of deep neural networks relates to increasing the learning capacity of a model (i.e. the ability to approximate more complex functions) by stacking multiple layers together, we need to ensure that our "deep" networks do not reduce to shallow ones. This crucial requirement is satisfied by the adoption of non-linear activation functions in the hidden layers. According to the way these functions are defined, we can further split the family of non-linear activations into: (i) *purely non-linear* and (ii) *piece-wise linear* functions.



(a) sigmoid function (b) tanh function

Figure 3.8: Sigmoidal (purely non-linear) activation functions.

Purely non-linear functions are generally defined from either trigonometric or exponential functions, and their behaviour can thus be highly non-linear when combined together (e.g. multiple layers). In practise, the most common examples of purely non-linear functions are:

- the **sigmoid** function, which squashes a real-valued number into range between 0 (the neuron does not fire at all) and 1 (the neuron has maximum firing). Therefore, because of this compelling biological interpretation, it has been extensively used in the past. The sigmoid activation function is element-wise and can be defined as follows:

$$\sigma(a^{(t)}) = \frac{1}{1+\exp(-a^{(t)})} \tag{3.14}$$

  As can be seen from Figure 3.8a, this function is steep for input values ranging from -2 and 2, thus any small changes in that region will cause the outputs to change significantly. The advantages (denoted by the $+$ symbol) and main drawback (denoted by the $-$ symbol) of the sigmoid activation function are summarised below.

  (+) As this function tends to bring its values to either end of the curve, this is desirable for the output units of a binary classifier, making clear distinction on prediction.

  (+) Unlike linear functions, the activations are bound in the range $(0,1)$ and this property is generally desirable because it will not saturate the activations.

  (-) On the other hand, training with gradient descent can be detrimental because sigmoid units are known to saturate and "kill" gradients. More precisely, when the neuron's activation saturates at either 0 or 1, the gradient at these regions is almost zero and the network will barely learn (almost no signal will flow through the neuron to its weights). For this reason, it is also important to carefully initialise the weights of sigmoid neurons in order to prevent saturation.

- the **softmax** activation function is the generalisation of the sigmoid function. In fact, whereas a sigmoid unit can represent a probability distribution over a binary variable, a softmax neuron is used to represent the probability distribution over different classes. Therefore, in light of this feature, the softmax function is typically used as the activation function in the output layer of a classifier, when more than 2 distinct classes need to be predicted. Similarly to the sigmoid, the softmax activation function is applied element-wise to its input $a^{(t)}$ (i.e. to each $a_i^{(t)}$ with $1 \leq i \leq K$ where K is the dimension of vector $a^{(t)}$) but we now need to ensure that the sum of the corresponding values is equal to 1. The function is thus formally defined below.

$$\text{softmax}(a^{(t)})_i = \frac{\exp(a_i^{(t)})}{\sum_j \exp(a_j^{(t)})} \tag{3.15}$$

  As the function provides a generalisation of the sigmoid, even softmax neurons can saturate, with the consequent detrimental effects on gradients. In particular, the softmax function saturates when the differences between the input values become extreme.

- the **tanh** (hyperbolic tangent) function can be considered as a scaled sigmoid function, so their characteristics are rather similar. As we can see from 3.8b, the tanh squashes a real-valued number to the range $[-1, 1]$ and it is defined as follows:

$$\tanh(a^{(t)}) = \frac{2}{1+\exp(-2a^{(t)})} = 2\,\sigma(2\,a^{(t)}) - 1 \tag{3.16}$$

  Again, because of the relationships with the sigmoid neuron, its activations can still saturate. However, the tanh activations are zero-centered (which is a desirable property for gradient descent) and the gradient is stronger than sigmoid (derivatives are steeper). For these reasons, the tanh non-linearity is generally preferred to that of the sigmoid.

On the other hand, piece-wise linear functions can be defined as real-valued functions whose graph is composed of straight-line sections, each of which is an affine function. Besides their deceiving name, it is worth remarking that the piece-wise functions presented in this section are still non-linear, thus their combination cannot be expressed by a shallow network. The most common activation functions in this category are described as follows:

- **Rectified Linear Unit** (ReLU), which makes the neuron non-active in case of non-positive input, otherwise it behaves like the identity function for every positive input. More formally, a ReLU unit computes the following activation function element-wise:

$$\text{relu}(a^{(t)}) = \max(0, a^{(t)}) \tag{3.17}$$

Since it introduction, the ReLU has been extensively used in deep neural network architectures; we can briefly outline its pros and cons as follows.

(+) The ReLU function is much less computationally expensive than *tanh* and *sigmoid* because it involves simpler mathematical operations.

(+) Any function can be effectively approximated with combinations of multiple ReLU functions (i.e. ReLUs as universal function approximator).

(+) Krizhevsky, Sutskever, and Hinton (2012) found that it can accelerate the convergence of stochastic gradient descent compared to the *sigmoid* and *tanh* functions.

(-) Since the range of this function is $[0, +\infty)$, activations are not bound and they can technically saturate (which might be undesirable for certain tasks).

(-) However, the main problem of ReLU units is due to the rigid thresholding which forces activations to zero whenever their input is negative. Units are thus fragile to large gradients which could cause the weights to update in such a way that the neuron will never activate. As a consequence, the gradient flowing through the unit will forever be zero from that point on. When this happens, we say that the ReLU units die during training because they will stop responding to variations in error and input. This phenomenon is called *the dying ReLU problem*.

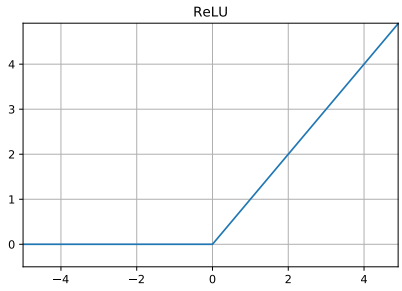
- **Leaky ReLU**, which is essentially a simple variation of the ReLU specifically designed to mitigate the dying ReLU problem presented before. The basic intuition is to give a small negative slope to the function in case of negative inputs, instead of just being zero. A Leaky ReLU unit applies the following function element-wise:

$$\text{lrelu}(a^{(t)}) = \max(a^{(t)}, \alpha a^{(t)}) \tag{3.18}$$

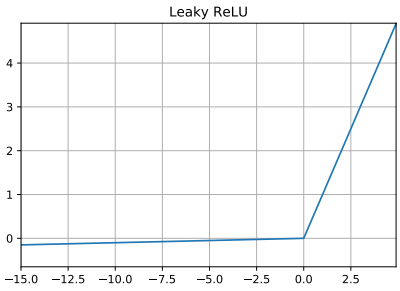
where α is a small constant, i.e. the slope of the function in the negative region. Several other variations of the ReLU exist and they all share the basic idea of letting the gradient be non-zero and eventually recover during training.

Activation functions in the context of RNNs

Now that the most common activation functions have been presented, we can focus on the task of choosing the right neuronal units for our RNN. First of all, the choice of neuronal units



(a) ReLU function



(b) Leaky ReLU with $\alpha = 0.01$

deserves a separate consideration since we need to make distinction among hidden and output units. This general consideration applies to any artificial neural network.

Choosing an activation function for the output units strictly depends on the loss function that our network will attempt to minimise, which in turn depends on the learning task. In this context, dealing with RNNs is no different than other families of neural networks. For instance, if the goal is to train an RNN for a binary classification problem (e.g sentiment analysis), we shall consider using sigmoid output units. Conversely, in case of image captioning, each output unit should model a probability distribution over words at a time; in this case, considering the multi-class nature of the classification problem, using softmax output units is more indicated.

The choice of hidden units is instead more challenging and it sometimes emerges from a trial and error approach: we start with an intuition of which hidden unit may be suitable for our task, then we train a network with such hidden units and evaluate its performance on a validation set. The design of hidden units is indeed an active area of research and does not yet have many definitive guiding theoretical principles (Goodfellow et al., 2016).

Nevertheless, although other deep neural networks like CNNs and GANs make extensive use of piece-wise linear activation functions, RNNs have additional requirements jeopardising their use in favour of purely non-linear units (despite the drawbacks of saturation). For instance, assuming that the RNN is randomly initialised, the recurrent connections would make the activations blow up after a few steps because of the unbounded activations of piece-wise linear functions. Hence, even though the first implementation of recurrent networks used *tanh* hidden units, it is still difficult to find a single activation function, among those presented in this section, which can effectively encode temporal information in a consistent long-termed way.

Intuitively, ReLUs seem inappropriate for RNNs because they can have very large outputs which in turn would be far more likely to explode than units that have bounded values. This observation is not always true, as some works investigated how ReLUs can be made to work well in RNNs and whether the ease of optimising them in feed-forward nets transfers to RNNs.

Among these studies, Le, Jaitly, and Hinton (2015) surprisingly demonstrated that, with the

right initialisation of weights, RNNs with ReLU activations in the hidden layer are relatively easy to train and their ability to model long-range dependencies is comparable to that of LSTMs. The approach followed by the authors is rather simple and intuitive as it consists of initialising the recurrent weight matrix to be the identity matrix and biases to be zero. The resulting model is called Identity-RNN (IRNN) and it has the following desirable implications:

- each new hidden state vector is obtained by simply copying the previous hidden vector then adding on the effect of the current inputs and replacing all negative states by zero. In the absence of an input, an IRNN thus remains in the same state indefinitely.

- when the error derivatives for the hidden units are back-propagated through time they remain constant provided no extra error-derivatives are added. This behaviour is similar to that of LSTMs when their forget gates are set so that there is no decay in the memory cell and makes it easier to learn long-term dependencies.

To empirically demonstrate the capabilities of IRNNs, (Le et al., 2015) tested them on four different benchmark experiments: two synthetic problems involving long-range temporal dependencies, a language modelling task and a speech recognition problem (most of these benchmark problems for RNNs are properly described in Section 3.4.2). From these experiments, not only IRNNs outperform RNNs with tanh units (i.e. vanilla RNNs) and ReLUs with random Gaussian initialisation, but also show comparable performances to LSTMs.

The authors also observed that for those tasks exhibiting less long-range dependencies, scaling the identity matrix by a small scalar is an effective mechanism to forget long-term information. This is helpful for speech-recognition, where similar inputs are provided to the network in neighbouring frames: taking short-term dependencies into account is as important as capturing long-range ones in this context. For this task, instead of the identity initialisation for the IRNNs matrices, Le et al. (2015) used $0.01 \odot I$. By scaling the identity matrix to more easily capture short-term dependencies, the resulting model is called iRNN. In contrast, initialising with the full identity (i.e. $i = 1$) tends to slow convergence and worsens results for speech recognition.

3.2.2 Choosing a loss function

As introduced in Section 3.1.1, if we denote the output of a recurrent network at time t and the corresponding target as $y^{(t)}$ and $\hat{y}^{(t)}$ respectively, the total loss for an input sequence x w.r.t. an output sequence y would be the accumulation of the losses over all the time steps:

$$\mathcal{L}\left(\{x^{(1)},\ldots,x^{(\tau)}\},\{y^{(1)},\ldots,y^{(\tau)}\}\right) = \sum_t L^{(t)} \tag{3.19}$$

Therefore, one of the most crucial part in defining a neural network, regardless of its specific type, consists in the selection or design of a proper loss function which can effectively capture the inconsistency between predicted value $y^{(t)}$ and its actual target $\hat{y}^{(t)}$.

To fully understand the role of loss functions and their relationship w.r.t the learning task, it is essential to present the following concepts. First, a loss function always returns a non-negative value, thus the robustness of our model increases along with the decrease of the value of the loss function. A loss function is also a significant component of the *structural risk function* of a model, which provides a broader context for the whole learning process as it describes the whole optimisation problem behind network's training. In particular, the structural risk function $J(\theta)$ consists of an *empirical risk term* (the loss function) and a *regularisation term*. Therefore, the mathematical formulation of the resulting optimisation problem can be defined as follows:

$$\begin{aligned} \theta^* = \arg\min_{\theta} J(\theta) = \mathcal{L}(\theta, x) + \lambda\Phi(\theta) = \arg\min_{\theta} \frac{1}{\tau}\sum_{t=1}^{\tau} L\big(y^{(t)}, \hat{y}^{(t)}\big) + \lambda\Phi(\theta) \\ = \arg\min_{\theta} \frac{1}{\tau}\sum_{t=1}^{\tau} L\big(y^{(t)}, f(x^{(t)}, \theta)\big) + \lambda\Phi(\theta) \end{aligned} \tag{3.20}$$

where $\Phi(\theta)$ is the regularisation or penalty term needed to prevent overfitting and promote generalisation, θ is the set of all of the model's parameters (essentially the weights) to be learned during training, and $f()$ represents the function modelled by a recurrent neural network. As can be seen, the definition of structural risk in Equation 3.20 is only valid for a single sequence $x = \{x^{(1)}, \ldots, x^{(\tau)}\}$, but it can be easily generalised for a set of sequences.

Maximum Likelihood Estimation

The description of the optimisation problem we are attempting to solve during training, given in Equation 3.20, only provides a practical viewpoint of the training process. A more formal analysis is now reported to emphasise the relationship between the nature of the function $f()$ modelled by our recurrent network and the true distribution of the data we are dealing with.

First, rather than approximating that some function could be a good estimator and then analysing its bias (i.e. evaluating the loss on the training set) and variance (i.e. how the loss differs when evaluated on the validation set), we look into how some principles can help derive specific functions that are good estimators for different models (Goodfellow et al., 2016).

For the moment, if we exclude the regularisation term in the structural risk function, we can introduce one of the most common of such principles, i.e. maximum likelihood estimation. To keep our notation consistent, we restrict our discussion to a target sequence of τ elements $y = \{y^{(1)}, \ldots, y^{(\tau)}\}$ drawn from the true but unknown data generating distribution $p_{\text{data}}(y)$. We

denote with $p_{\text{model}}(y;\theta)$ the family of probability distributions parameterised by θ. The maximum likelihood estimator for θ is thus defined as follows:

$$\theta_{ML} = \arg\max_{\theta} \prod_{t=1}^{\tau} p_{\text{model}}(y^{(t)};\theta) \tag{3.21}$$

As this product over many probabilities can be numerically unstable (e.g. it can lead to numerical underflow for long sequences) we can derive a more convenient but equivalent optimisation problem if we observe that taking the logarithm of the likelihood does not change its argmax, but does conveniently transform a product into a sum:

$$\theta_{ML} = \arg\max_{\theta} \sum_{t=1}^{\tau} \log p_{\text{model}}(y^{(t)};\theta) \tag{3.22}$$

To obtain a version of the maximum likelihood principle that is expressed as an expectation with respect to the empirical distribution $\hat{p}_{\text{data}}(y)$, we can observe that argmax does not change when we re-scale the cost function; thus, dividing by τ yields:

$$\theta_{ML} = \arg\max_{\theta} \mathbb{E}_{y\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(y;\theta) \tag{3.23}$$

We can interpret this formulation of the maximum likelihood principle by considering that our objective is to minimise the dissimilarity between the empirical distribution $\hat{p}_{\text{data}}(y)$ defined by the training set (which provides an approximation of $p_{\text{data}}(y)$) and the model distribution. The degree of dissimilarity between the two distributions can be measured by the Kullback-Leibler (KL) divergence. The KL divergence is indeed defined as a measure of how one probability distribution (that of our model) diverges from a second probability distribution (that of the training data). Despite the fact that the KL divergence cannot be formally considered as a distance metric between probability distributions, because it is not symmetric, we can still appreciate its formulation as another way to interpret the maximum likelihood principle. In fact, the KL divergence is given by the following equation:

$$D_{KL}(\hat{p}_{data}||p_{model}) = \mathbb{E}_{y\sim\hat{p}_{\text{data}}}[\log \hat{p}_{data}(y) - \log p_{\text{model}}(y)] \tag{3.24}$$

where the term on the left is known as *entropy* of the empirical distribution, i.e. the average amount of information produced by the distribution of the training data; and the other term is the *cross-entropy* of the two distributions. In other words, the KL divergence measures the likelihood that samples represented by the empirical distribution described by the training data were generated by the model's distribution. Therefore, a KL divergence close to 0 indicates that our model is good in approximating the empirical data distribution; if even equal to 0 we are guaranteed that the distribution of our model was generated by the empirical distribution.

Conversely, a KL divergence tending to infinity implies that the two distributions have no similarities. In the latter case, two interpretations are possible: either the family of distributions chosen for our model does not resemble that of the empirical data, or the parameters of our model are such that we are still underfitting our data.

As observable, the term on the left in Equation 3.24 (i.e. the entropy of the empirical distribution) is constant during training; in other words, it is a function only of the data generating process, not the model. This observation implies that, in regard to the maximum likelihood principle, when we train the model to minimise the KL divergence, we only need to minimise the cross-entropy (i.e. the negative log-likelihood) between the two distributions:

$$-\mathbb{E}_{y\sim\hat{p}_{\text{data}}}[\log p_{\text{model}}(y)] \tag{3.25}$$

Therefore, since minimising the KL divergence and minimising the cross entropy between the two distributions can be considered as two faces of the same coin, Equations 3.23 and 3.25 refer to the very same problem. In fact, many authors use the term "cross-entropy" to identify specifically the negative log-likelihood of a Bernoulli distribution, but (Goodfellow et al., 2016) are against this view and consider it as a misnomer.

More generally, any loss function consisting of a negative log-likelihood (NLL) is a cross-entropy between the empirical distribution defined by the training set and the probability distribution defined by our model. For instance, as we will see in the next section, mean squared error is the cross-entropy between the empirical distribution and a Gaussian model.

To conclude, whereas the optimal parameters θ^* of our model's distribution are the same regardless of whether we are maximising the likelihood or minimising the KL divergence, the values of the objective functions are different. On a practical point of view, we often phrase both as minimising a cost function, and maximum likelihood thus becomes minimisation of the NLL, or equivalently, minimisation of the cross entropy. This perspective can be helpful as the KL divergence has a known minimum value of zero.

A taxonomy of the most common loss functions

Now that the objectives of the optimisation problem are clear, together with the principle of maximum likelihood estimation, this section focuses on the most common families of loss functions used to compute the empirical risk term in Equation 3.20; we leave the discussion of the regularisation term for the next section.

$$\mathcal{L}(\theta,x) = \frac{1}{\tau}\sum_{t=1}^{\tau} L\big(y^{(t)}, f(x^{(t)},\theta)\big) \tag{3.26}$$

Different loss functions will give different measures of error for the same prediction, and the

selection/design of the loss function to adopt in our model is strictly dependent on the learning task. More precisely, we have several loss functions for different learning tasks, where the choice relies on the true distribution behind the data we are dealing with.

Generally, despite the intrinsic diversity of such learning tasks, most loss functions can be seen as particular cases of others. More precisely, we argue that the majority of tasks can be modelled in terms of either a classification or a regression task. For instance:

- the task of *anomaly detection*, in its easiest form, can be considered as a special case of binary classification: at each time step we need to decide whether the behaviour of the system under analysis is regular or not;
- in *polyphonic music generation* we are still dealing with a classification problem (but now with more classes): at each time step, the model predicts the most likely set of notes given the previous ones and the hidden state of the network at the previous time step;
- testing the ability of a network on synthetic *algorithmic problems* (e.g. the adding problem), can be considered as a regression problem: at each time step, the model computes a real number that should be as close as possible to the partial output of the algorithm we are trying to learn, at the current point in the sequence.

Therefore, instead of defining different families of functions for different learning tasks, we just need to consider the two families of regression and classification loss functions.

Regression loss functions

The loss functions presented in this section are typically used for regression problems, i.e. when the target variable to predict is continuous. Nonetheless, these functions are not only limited to regression problems. In fact, the empirical study by Janocha and Czarnecki (2017) indicated that some of these functions can also provide probabilistic interpretation (in terms of expected misclassification), and can thus be used for classification tasks.

- **Mean Squared Error** (MSE) is a quadratic loss function that has been extensively used as the standard performance measure in linear regression. In fact, in the case of regression, the probabilistic component of the model is generally assumed to be Gaussian (Jordan & Jacobs, 1994) and the cross-entropy between the empirical distribution and a Gaussian model naturally results in the mean squared error loss function.

  In practice, the method of minimising the MSE is called Ordinary Least Squares (OLS), i.e. the optimised fitting line should be a line minimising the sum of distances of each point to the regression line. The standard form of the MSE loss is defined as follows:

$$\mathcal{L}_{MSE} = \frac{1}{\tau}\sum_{t=1}^{\tau}(y^{(t)} - \hat{y}^{(t)})^2 \tag{3.27}$$

However, it is worth observing that using sigmoid output units, the mean squared error would suffer from slow convergence when the difference between predicted value and true label is large; this behaviour can be easily monitored by computing the gradient of the loss function w.r.t. the weights of the sigmoidal output units.

- **Mean Absolute Error** (MAE) is a loss function similar to MSE and is used to measure how close model's predictions are to the eventual outcomes in absolute terms. The mathematical definition of this loss function is given as follows:

$$\mathcal{L}_{MAE} = \frac{1}{\tau}\sum_{t=1}^{\tau}\left|y^{(t)} - \hat{y}^{(t)}\right| \tag{3.28}$$

Despite their similarities, there are some major differences between MSE and MAE which are worth considering. On one hand, the gradient of MSE is easier to compute, whereas MAE requires more complicated tools for this purpose (e.g. linear programming). On the other hand, because of the squared differences in its definition, MSE gives greater influence to large errors than smaller ones. This property might be desirable when the number of outliers is rather limited in the training set, otherwise the learning task would considerably be affected by such discrepancies.

On the other hand, MAE is more robust to outliers due to the lack of a squared term, and it uses the same scale as the data being measured (it is also known as a scale-dependent accuracy measure). For these characteristics, mean absolute value is a common measure of forecast error in time series analysis, even though it cannot be used to make comparisons between series using different scales. However, minimising the MAE loss function does not correspond to maximising the likelihood of Gaussian random variables, and this is one of the reasons why its application is limited to specific domains.

- The **L1** loss function is simply defined as the sum of absolute errors of the difference between actual value and predicted value. L1 is mathematically similar to MAE, as the only difference consists in the absence of the division by the sequence length τ. Analogously, the **L2** loss function is based on the same principle but applied on MSE. Both these functions are respectively defined by the following equations.

$$\mathcal{L}_1 = \sum_{t=1}^{\tau}\left|y^{(t)} - \hat{y}^{(t)}\right| \tag{3.29}$$

$$\mathcal{L}_2 = \sum_{t=1}^{\tau}(y^{(t)} - \hat{y}^{(t)})^2 \tag{3.30}$$

As better detailed in the following section, L1 is often used as an auxiliary term in the loss function to promote sparseness of the learned representations. Similarly, L2 is sometimes applied to weights in order to prevent them from growing to infinity.

As mentioned before, Janocha and Czarnecki (2017) showed how these loss functions, despite their regressive nature, can provide reasonable probabilistic interpretation for classification and can thus be used as a main classification objective.

Classification loss functions

The output variable in a classification problem is generally a probability value $f(x^{(t)})$ called the score for the input $x^{(t)}$. For instance, if $x^{(t)}$ represents a musical chord at time t, then $\hat{y}^{(t)}$ would correspond to the conditional probability distribution over the possible chords for the next time step, given the current chord (and the state of our model at the previous time step). Therefore, the magnitude of the score represents the confidence of our prediction. The most common loss functions for classification problems are reported below.

- **Binary cross entropy** is the preferred loss function for binary classification problems. In these problems, the output is a discrete random variable having possible outcomes of "failure" or "success". As the random variable y is binary, the probabilistic component of the model is assumed to be the Bernoulli distribution (Jordan & Jacobs, 1994). Formally, binary cross entropy is defined as follows:

$$\mathcal{L}_{CE} = -\frac{1}{\tau}\sum_{t=1}^{\tau}\left[y^{(t)}\log(\hat{y}^{(t)}) + (1-y^{(t)})\log(1-\hat{y}^{(t)})\right] \tag{3.31}$$

  As can be noted from this equation, cross entropy measures the divergence between two probability distributions: if it is large, then the difference between two distributions is large; if small, then we can conclude that the two distributions are similar to each other. Training a recurrent network with cross entropy consists in finding the optimal set of parameters θ^* to bring the probability distribution of our model as close as possible to the training data distribution, thus minimising their cross entropy.

  The cross entropy function is also used (implicitly) in character-level language models, where the objective is to approximate the probability distribution of the next character given the sequence of past characters (Graves, 2013). In this context, even though we often read that the **bits per character** (BPC) loss function is used for such tasks, we are still dealing with the average cross-entropy but used with logarithms in base 2.

- **Multi-class cross entropy** is the generalised version of binary cross entropy and can be considered as the application of the maximum likelihood principle when more than 2 classes need to be predicted. The probabilistic component of the model is assumed to be the Multinoulli distribution – generally referred to as *negative log likelihood* (we remind that binary cross-entropy is also a special case of NLL). In particular, whereas sigmoid output units are typically used with binary cross entropy, softmax output units (multi-class classification) are used in tandem with the general form of the negative log-likelihood loss function. As observed in the previous section, NLL is defined as follows.

$$\mathcal{L}_{NLL} = -\frac{1}{\tau}\sum_{t=1}^{\tau}\log(\hat{y}^{(t)}) \tag{3.32}$$

  The NLL loss function has been extensively in sequence modelling problems, including polyphonic music modelling (Allan & Williams, 2005), but also for other tasks such as handwritten character recognition (e.g. Sequential MNIST).

In addition to the former functions, providing the mathematical foundations for more complex losses, it is also common to use other functions for evaluation/interpretation purposes. An example is **perplexity**, a common evaluation measure for language models based on average probability (Shannon, 1948). Perplexity is defined as the exponential of the entropy of the model distribution (assuming that entropy has natural logarithm), and more formally as:

$$\mathcal{L}_{perp} = e^{\frac{1}{\tau}\Sigma_{t=1}^{\tau}\left(\hat{y}^{(t)}\log(\hat{y}^{(t)})\right)} \tag{3.33}$$

The interpretation of perplexity is rather intuitive, as it can be seen as the average of choices per event/token: the higher it is, the more choices our language model has, thus the more uncertain it is. In other words, perplexity can be considered to be a measure of, on average, how many different equally most probable words can follow any given token. Lower perplexities thus represent better language models. However, perplexity may be a poor performance measure for other tasks, such as *speech recognition*. This is due to its inability to account for the relevance of acoustically similar or dissimilar words.

3.2.3 Regularisation techniques

The previous subsection introduced the most common loss functions used to train ANNs, with a focus on their relationships with sequence learning tasks. As we observed in Equation 3.20, the loss function is only one of the terms in the structural risk function. Here, we review the missing component of this equation – the regularisation term $\Phi(\theta)$, and present the main regularisation techniques used in both the general context of ANNs and those more specific to RNNs.

To open this section, we start with the following question: *what is regularisation and why should we care about it?* Regularisation refers to the plethora of techniques and methods used in machine learning (including ANNs), and more generally in optimisation contexts, to limit their learning capacity and prevent overfitting. As we typically use a separate partition of the training data, to monitor our learning procedure (the validation set), overfitting occurs when the gap between the training and the validation losses starts increasing after a number of training epochs. In other words, if no regularisation method is applied, we would risk to end up with a model that is extremely good at fitting the training set but fails to generalise to new/unseen data.

In deep learning, most regularisation strategies are based on regularising estimators, by trading increased bias (higher training error) for reduced variance (lower validation error). An effective regularisation method is one that makes a profitable trade, reducing variance significantly while not overly increasing the bias (Goodfellow et al., 2016). Here, we can categorise regularisation methods into four families, according to their nature and the objectives of these approaches. In particular, we consider the following classes: *term-based regularisation*, *adding stochasticity*, *dataset augmentation* and *early stopping*.

Term-based regularisation

This class of regularisation methods is based on adding a regularisation term to the structural risk function, as summarised in Equation 3.34. As the term is part of the objective function, we are not only minimising for the loss, but also penalising certain parameter configurations that could potentially overfit the model. The general learning framework is defined as:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta, x) + \lambda\Phi(\theta) \tag{3.34}$$

where $\lambda \in [0, \infty)$ is a hyper-parameter weighting the relative contribution of the regularisation term. Different choices of the regularisation function $\Phi(\theta)$ would thus result in different solutions being favoured. In practice, it is preferable to use a regularisation function that penalises only the weights of the affine transformation at each layer, but not the biases. Therefore, we assume that the set of parameters θ to optimise corresponds to the set of ANN's weights.

The most common regularisation techniques used to that extent are summarised below.

- the $\mathbf{L_2}$ parameter regularisation, also known as weight decay and ridge regression, is probably the most used regularisation technique. It can be easily described as a kind of penalisation of the squared magnitude of all weights W in our neural network:

$$\Phi(\theta) = ||\theta||_2 = \sum_{j=1}^{|\theta|} W_j^2 \tag{3.35}$$

For mathematical convenience, it is also common to use a $1/2$ factor for the norm, as the gradient of that term w.r.t. the parameter w is just w instead of $2w$.

The intuitive interpretation of L_2 regularisation is that it consistently penalises peaky weight vectors, thereby preferring diffuse ones. In this way, we are actually encouraging the network to use all of its inputs rather than only some of them. Moreover, if we consider the contribution of the λ parameter, during the parameter update of gradient descent, L_2 forces every weight to decay linearly ($W = W - \lambda W$) towards 0.

- the $\mathbf{L_1}$ parameter regularisation is an alternative form of regularisation that performs the sum of absolute values of the individual weight parameters. More formally:

$$\Phi(\theta) = ||\theta||_1 = \sum_{j=1}^{|\theta|} |W_j| \tag{3.36}$$

Because of the non-squared absolute values, L_1 regularisation has the intuitive property that it leads the weight vectors to become sparse during optimisation. In this context, the concept of sparsity refers to the fact that some parameters have an optimal value of zero. Therefore, under this settings, neurons tend to indirectly perform a sort of feature selection process; in other words, they end up using only a sparse subset of their most important inputs – which might be beneficial in case of noisy inputs.

To summarise, whereas L_2 regularisation pushes weights to small but diffuse numbers, L_1 might cause parameters to be sparse, especially for large values of λ. Therefore, unless we are concerned with explicit feature selection due to noisy data, it is generally preferred to adopt L_2 regularisation. It is also possible to combine them, and this approach is known as *elastic network regularisation* (Zou & Hastie, 2005).

- the **activation stabilisation** method is a regularisation technique explicitly designed for recurrent networks. Introduced by (Krueger & Memisevic, 2015), the intuition behind this approach consists in bounding the difference between the activations of the hidden units at two consecutive time steps, i.e. $h^{(t-1)}$ and $h^{(t)}$. This results in the introduction of an additional regularisation term (with its own λ) defined as follows:

$$\Phi_h = \frac{1}{\tau} \sum_{t=1}^{\tau} (||h^{(t-1)}||_2 - ||h^{(t)}||_2)^2 \tag{3.37}$$

Such an additional term is meant to stabilise the norm of the hidden vector when generalising long-term dependencies. In fact, by limiting the exposition of the hidden activations

to much variation, we are indirectly forcing our recurrent units to retain the current context so as not to forget it too fast, but at least progressively.

Achieving regularisation by adding stochasticity

Adding non-deterministic behaviour to our model at training time is one of the possible ways to make the network more robust to diverse inputs and to increase its generalisation capabilities. More precisely, whereas term-based regularisation affects the parameter learning process only by means of a modification of the structural risk function, this other regularisation method is instead achieved by altering the specification of our network in such a way that the computation for a given input is not guaranteed to always produce the same result.

Nevertheless, the stochastically-altered model is only used during training, for the specific purpose of regularisation. Once the training process has finished, the correspondent ANN without the stochastic behaviour/component is evaluated on the test set and used for inference.

The most known ways to add stochasticity to a neural network are presented as follows.

- **Injecting noise** to the weights is one of the possible regularisation methods that introduces stochasticity directly in the hidden units. We can think of noise applied to the weights of a neural network as a form of regularisation encouraging stability of a function (Goodfellow et al., 2016). Technically, we assume that with each presentation of an input, we also include a random perturbation $\varepsilon_W \sim \mathcal{N}(\varepsilon; 0; \eta I)$ of the network weights.

  This method has been used especially in the context of recurrent networks (Graves, 2011) and it can be interpreted as a stochastic implementation of Bayesian inference over the weights. In particular, this alternative point of view would consider the weights of a network to be drawn from a probability distribution reflecting their uncertainty.

- **Dropout** is a simple and effective regularisation method that has increasingly become a gold standard since its introduction Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov (2014). At training time, dropout is implemented by just keeping a neuron active with some probability k or setting it to zero otherwise:

$$h^{(t)} = k \odot h^{(t)} + (1-k) \odot 0 \tag{3.38}$$

  The probability of leaving a neuron alive k is thus a binary vector mask which is multiplied element-wise to each activation. As k is not learned during training, this comes solely at the expense of introducing an additional hyper-parameter. On the other hand, dropout has a considerable advantage over other regularisation approaches, i.e. it can be thought of as a method of making bagging practical for ensembles of large deep neural networks.

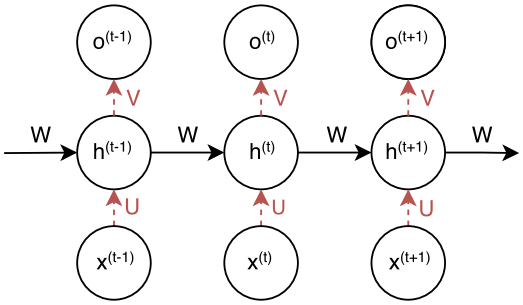


Figure 3.10: Schematic illustration of the dropout operator proposed in (Zaremba et al., 2014). The dashed red arrows indicate connections where dropout is applied; the solid black ones denote those where dropout is not applied.

In fact, as bagging consists in training several models so as to evaluate them on the test set, it seems unpractical to use such an approach for deep network because of the many possible architectural and non-architectural variations. Nevertheless, the behaviour of dropout resembles that of an ensemble consisting of all sub-networks that can be formed by removing non-output units from an underlying base network (Goodfellow et al., 2016).

However, Bayer et al. (2013) observed that the original implementation of dropout is detrimental for RNNs, as recurrence can considerably amplify noise, which in turn impact on learning. To address this, Zaremba, Sutskever, and Vinyals (2014) introduced a variation of the dropout operator that isolates the stochastic behaviour to the non-recurrent connections. The resulting operator (Figure 3.10) still corrupts the information carried by units, forcing them to perform their intermediate computations more robustly.

Another notable dropout implementation specifically for RNNs is RNNDrop (Moon, Choi, Lee, & Song, 2015). In short, this variant manages to apply dropout to both recurrent and non-recurrent connections, but a single dropout mask is generated randomly for each training sequence and fixed throughout the sequence.

- **Zoneout** can be considered as an alternative to dropout which can be applied through time for any RNN architecture. Whereas dropout injects noise by shutting the activations of some units, zoneout randomly replaces some units' activations with their activations from the previous time step (Krueger et al., 2016). More formally, zoneout uses a Bernoulli mask k to affect the activations flow as follows:

$$h^{(t)} = k \odot h^{(t)} + (1 - k) \odot 1 \tag{3.39}$$

Therefore, compared with dropout, zoneout preserves information flow forwards and backwards through the network, which is also a desirable property for mitigating the

vanishing gradient problem. In their work, Krueger et al. (2016) empirically demonstrate that zoneout outperforms many alternative regularisers to achieve results competitive with the tate of the art on several sequence modelling tasks.

Dataset augmentation

The availability of large amount of data, together with the increased computational capabilities of today's hardware, have both ignited the effectiveness and the popularity of deep learning. Assuming that the problem/task is well-formulated, the recipe is always the same: the larger our training data, the better our neural network is expected to perform and generalise – a single expedient that can potentially improve our model in terms of reduced bias and variance.

Nevertheless, in some domains or for some specific tasks, the availability of data can be fairly limited. This is especially true for for all those supervised problems where collecting and labelling new data is an expensive process, in terms of time, expertise, money, accessibility, and so forth. A simple yet effective way to mitigate this problem is to create new data from the existent sample, so as to extend our dataset – a process known as *data augmentation*.

Data augmentation can be defined as the synthetic generation of novel, yet meaningful, data through the introduction of new samples created by the perturbation of the original dataset, while preserving the label of the original (mutated) samples (Bloice, Stocker, & Holzinger, 2017). If properly designed, this simple approach can let the model fit the data more robustly so as to be invariant to all those transformations/perturbations applied to the original samples.

Depending on the task, the literature provides several data augmentation operators and techniques, which are also implemented in most deep learning frameworks – especially for computer vision. For instance, in *object recognition* it is intuitive to observe that the high dimensionality of images makes it possible to simulate several factors of variations which leave the actual content/object represented in the image semantically intact. As argued in (Goodfellow et al., 2016), operations like translating the training images a few pixels in each direction can improve generalisation, even if the model has already been designed to be translation invariant (e.g. Convolutional NNs). Other operations such as rotating, scaling and applying colour filters to the image have also proven to be quite effective in this domain. Overall, any mutation operator is plausible as long as the resulting transformations do not change the correct class/label of the original samples. For instance, in character recognition, horizontal flips (180° rotations) are not applicable as the differences between 'b' and 'd', and between '6' and '9' would be lost.

In sequence learning, dataset augmentation is effective for fewer tasks, such as speech recognition and for sequential image processing. The general procedure for the former task is injecting random noise to the input audio signals so as to make the model invariant to a possible disturbance or interference, which can also emerge in the real world. We can also think of

Figure 3.11: Examples of dataset augmentation for object recognition

changing the pitch and intonation of the signal in order to generalise to the more-personal characteristics of voice, and achieve better performance on data originating from unseen speakers.

However, we can find at least two sequence learning tasks in which the application of data augmentation is even hard to conceive. More precisely, we can observe that:

- in *language modelling* researchers have attempted augmentation operators like word-swapping, syntax tree manipulations and adversarial networks. Nevertheless, such approaches can be extremely computationally-intensive and there is a strong limit to how much generalisation we can obtain from these methods.

- in *time series forecasting*, the limited availability of observations and the actual nature of the data being modelled make it difficult to design data augmentation operators other than noise injection, which is still not enough to obtain novel observations that our network could potentially exploit. For instance, in earthquake prediction, it is difficult to imagine plausible and meaningful ways to augment the time series of seismic events.

Early stopping

If we apply all the regularisation techniques discussed so far, we are still not guaranteed to end up with a model with no variance. In fact, when training a large model with enough representational capacity to potentially overfit the training data, we often observe that training error decreases steadily over time, but validation set error begins to rise again. This is almost inevitable, and it generally occurs at some point during training (Figure 3.12).

Nevertheless, by monitoring the training process and watching both training and validation losses, we can obtain a model with low variance by returning to the parameter setting at the point in time with the lowest validation set error (Goodfellow et al., 2016). This is the idea behind the early stopping strategy, which can be easily implemented by considering these steps:

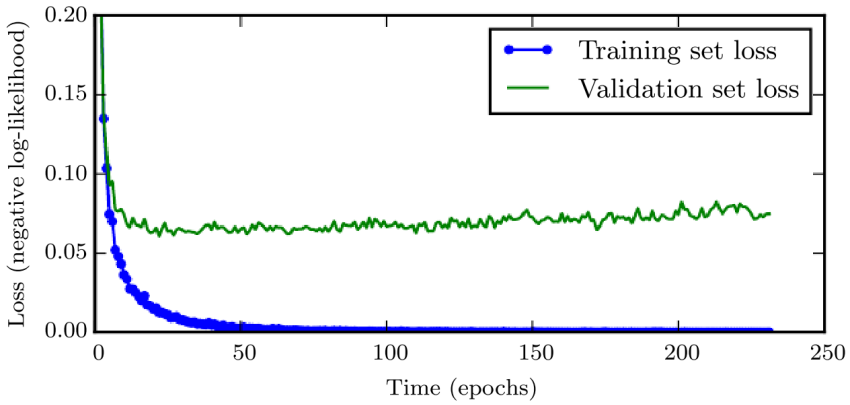


Figure 3.12: Example of training and validation losses (NLL). Although the training loss decreases consistently over time, the validation error eventually begins to increase again, forming an asymmetric U-shaped curve. Source (Goodfellow et al., 2016).

1. every time the validation error improves, we store a copy of the model parameters;

2. when training terminates, we return these parameters, rather than the latest ones.

Early stopping is one of the most common regularisation technique deep learning. From a technical viewpoint, its effectiveness is also attributable to the following considerations.

- It is simple to implement and generally affordable in terms of computational demand. More in detail, early stopping requires running the validation set evaluation periodically during training and maintaining a copy of the best parameters over the epochs. The first requirement can be addressed, to some degree, by running training and validation in parallel on a separate machines or processing units. On the other hand, the cost of storing the current best parameters is generally negligible, as these can be stored in a slower and larger form of memory.

- Compared to other methods, early stopping does not require almost any change in the underlying training procedure, the objective function, or the set of allowable parameter values. It is therefore an unobtrusive form of regularisation. This observation is not trivial, since other forms of regularisation need to be carefully tuned in order to avoid underfitting regimes. For instance, when using weight decay we should be careful not to excessively penalise our parameters, as this can potentially "keep the model stuck" in a bad local minimum (a solution with tiny weights).

- It reduces the computational cost of the training procedure because it limits the number of training iterations needed to reach convergence without overfitting the training data.

An alternative way to interpret early stopping, given in (Goodfellow et al., 2016), is that of an efficient hyper-parameter selection algorithm. Indeed, if we think of the number of training epochs as just another hyper-parameter, early stopping would allow us to control the effective capacity of the model by determining how many steps it can take to fit the training set.

However, as better outlined in Section 3.3.2, most hyper-parameters are expensive to tune properly because they need to be fixed before training and their effect can only be evaluated after a number of epochs. In contrast, the "training time" hyper-parameter offered by early stopping is unique in that because a single run of training tries out many values for it.

3.3 Training recurrent neural networks

Training an artificial neural network, regardless of its specific type, corresponds to the optimisation problem of finding the optimal set of the network's parameters that minimise the structural risk function (Section 3.2.2). In the machine learning literature we can find several training algorithms and methods, some of which are of general applicability while others show properties that are more suitable for specific kinds of architectures. Once the training process is over, thus we have found a set of parameters leading the structural risk function to a local optimum, we need to measure the performance of our model on the test set: evaluating the structural risk function on the test set would then provide a generalised measure of the learning capabilities of the model on the specific learning task for which it was designed.

Besides the *learnable parameters*, such as the weights of our model, there is also another important category of parameters, such as the learning rate and the regularisation strength, which are chosen before and kept fixed throughout training. These are known as *hyper-parameters*, and their selection is another crucial task that is addressed by separate strategies.

The scope of this section is to expand the first two topics, i.e. training techniques and hyper-parameter selection, with particular emphasis on their application in the context of RNNs.

3.3.1 Gradient-based learning with BPTT

Although we can find other optimisation algorithms for training recurrent neural networks, such as Hessian free (HF) optimisation (Martens & Sutskever, 2011), second order methods (Mirikitani & Nikolaev, 2010), extended Kalman filter learning (Williams, 1992), and global optimisation (Angeline, Saunders, & Pollack, 1994), the gold standard technique for such a task remains the family of *gradient-based learning*. One of the possible reasons behind this global trend is the high computational complexity of the before mentioned alternatives; in addition, some of them can be unstable and not scalable enough under certain configurations. On the

other hand, the simplicity of gradient-based learning algorithms, together with their reasonable computational requirements, make them the most preferred family of optimisation techniques for training RNNs and artificial neural networks in general.

Gradient descent is a robust optimisation method where the gradient of the structural risk function $J(\theta)$ (also known as *cost function*) with respect to the parameters of our model θ is computed $\nabla_\theta J(\theta)$, and a portion η of that gradient is subtracted off of the parameters (Dozat, 2016). The value η is generally referred to as *learning rate* – indicating the size of the steps we take to reach a local minimum of the cost function. In other words, parameters are updated in the opposite direction of the gradient. As described by (Ruder, 2016), we follow the direction of the slope of the surface created by the cost function downhill until we reach a valley.

As we can observe, gradient descent describes an approach for learning the parameters of our model, rather than a specific procedure. The three main algorithms implementing this approach, which also provide a baseline for further improvement, are outlined as follows.

- **Batch gradient descent** (BGD), also known as vanilla gradient descent, computes the gradient of the cost function w.r.t. the parameters for the entire training set:

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{3.40}$$

  Using BGD, we update the parameters of our model only when the loss function has been evaluated on the whole training set, so as to compute its gradient w.r.t. our parameter. The following two major drawbacks thus emerge from this consideration: (i) excessive memory requirements and potential intractability of the algorithm in case of large datasets; (ii) inability to be applied in online application contexts, i.e. on-the-fly learning.

  On the other hand, parameter updates are more consistent than those of the other algorithms reported below. Therefore BGD is guaranteed to converge to the global minimum for convex error surfaces and to local minimum for non-convex surfaces (Ruder, 2016).

- **Stochastic gradient descent** (SGD) computes the gradient of the cost function w.r.t. the parameters using a single training example $x^{(i)}$ with its label $\hat{y}^{(i)}$ as follows:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{3.41}$$

  where i ranges over sequences of the training set and not over elements of a sequence.

  The intuition behind SGD is that of approximating the true gradient of the cost function (as done in vanilla gradient descent) by a gradient at a single example. In this way, the

algorithm performs one update at a time and it is therefore faster than BGD, as computation time per update does not grow with the number of training examples. SGD is thus suitable for online learning and can converge within some fixed tolerance of its final test error before it has processed the entire training set.

Although BGD is guaranteed to converge to the minimum of the basin the parameters are placed in, updating the model's parameter with an approximated gradient at each training example causes inevitable fluctuations of the cost function. On one hand, this behaviour is desirable for the sake of a better exploration of the problem landscape, which in turn increases the likelihood of finding better local minima. On the other hand, trading exploitation for increased exploration might jeopardise convergence to the exact minimum. For this reason, the learning rate is typically much smaller than in BGD; and a common approach is to decrease it progressively according to a schedule. With these revisions, SGD shows the same convergence behaviour as BGD, almost certainly converging to a local minimum for non-convex and convex optimisation respectively (Ruder, 2016).

To conclude, as we update our parameters once a single training sample is processed, the learning process would be implicitly affected by the order in which we present the data to the algorithm and lead to poor convergence. To address this issue, a simple but effective method consists in randomly shuffling data prior to each epoch of training.

- **Mini-batch gradient descent** (MBGD) is probably the most common gradient-based algorithm in deep learning, as it seeks to find a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent. Intuitively, MBGD divides the training set into small partitions (i.e. *mini-batches*) that are used to compute the approximated gradient of the cost function w.r.t. the model parameters so as to perform an update for every mini-batch of n training examples:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{3.42}$$

Implementations may choose to sum the gradient over the mini-batch or take the average of the gradient, which further reduces the variance of the gradient.

As mini-batch gradient descent takes the best of both BGD and SGD, the advantage of this learning method is twofold: (i) reduction of the variance during the updates of parameters, with the consequent effect of a more stable convergence; (ii) exploitation of parallel hardware architectures (e.g. graphical processing units) by means of the optimised matrix operations offered by today's deep learning frameworks.

A proper choice of the mini-batch size n permits the leveraging of the amount of exploitation and exploitation of the learning process: small values of n give a learning process that converges quickly at the cost of noise in the training process; large values give a learning process that converges slowly with accurate estimates of the error gradient. Common mini-batch sizes range between 32 and 256, and the modeller should take into account both the requirements and the eventual limitations of the learning problem as well as the capabilities of the hardware available. Indeed, n is just another hyper-parameter.

The gradient-based methods reported before are of general applicability and are not limited to ANNs. What characterises learning in the context of neural networks is the use of *back-propagation* – computing the gradient of the cost function w.r.t. every single parameter that is part of the model. Whereas in *forward-propagation* we compute the output of our network by propagating inputs up to the hidden units, during *back-propagation* the information from the cost flows backward through the network in order to compute the gradient. This last step is so relevant for the learning process as it allows to measure the contribution of each single parameter to the cost function (the gradient). We can picture this process as "blaming" each parameter for the actual extent of its responsibility for having mistakenly performed the forward pass.

Back-propagation thus refers only to the method for computing the gradient, while the gradient-based algorithms outlined before "consume" that information in order to update the parameters of the model. Even though it is out of the scope of this chapter to provide a description of the back-propagation (BP) algorithm, it is still worth remarking that recurrent neural networks cannot be trained with the version of BP used for feed-forward networks.

Back-propagation through time (BPTT) is the most common variant of BP that has been explicitly designed for training RNNs. The idea consists in applying the generalised form of the back-propagation algorithm Goodfellow et al. (2016) to the unrolled computational graph: at each time step an input of the sequence is presented to the RNN and the corresponding output is computed (forward pass); once the current value of the cost function is computed, the network is then unrolled until the beginning of the so-far processed input sequence so as to compute the gradients backwards through time (backward pass); to conclude, the network is rolled back up and its parameters are updated accordingly with a gradient-based algorithm.

However, if we consider long sequences, BPTT would require a huge computational cost for a single parameter update. Intuitively, each time step of the unrolled RNN may be seen as an additional layer of a traditional feed-forward network. For instance, as outlined in (Sutskever, 2013), "*the gradient of a recurrent network on sequences of length* 1000 *costs the equivalent of a forward and a backward pass in a neural network that has* 1000 *layers*". A naive approach to reduce this cost might be that of splitting such long sequences into 50 sub-sequences of length 20, thus treating each sub-sequence as a training example on its own. Nevertheless, despite this

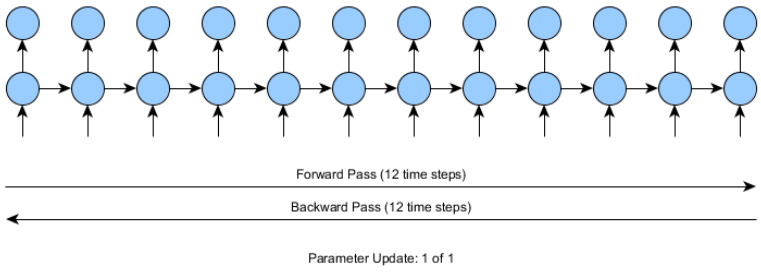


Figure 3.13: Example of a single BPTT update (from `deeplearning4j.org`).

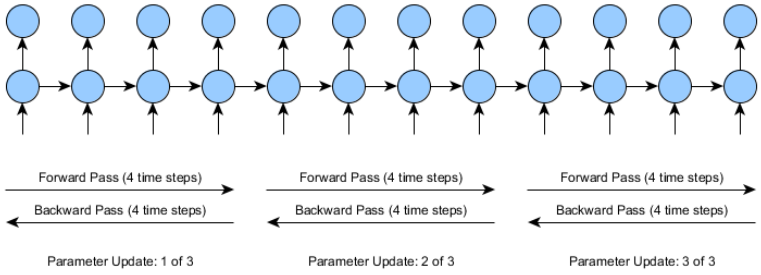


Figure 3.14: Analogous representation of TBPTT updates (from `deeplearning4j.org`).

approach may work in practice, our network would be completely unable to capture temporal dependencies spanning more than 20 time steps.

Truncated BPTT (TBPTT) is a modified version of the BPTT algorithm which has the same per-iteration cost of the naive approach, but it is more adept at exploiting temporal dependencies of longer range (Sutskever, 2013). In particular, TBPTT processes the sequence step by step and performs a standard BPTT update periodically (every k_1 time steps) but the computational graph is only unrolled for a fixed number of time steps (k_2 time steps); the parameter update can thus be computationally affordable if k_2 is small. As can be noticed, the hidden states of the recurrent network have been exposed to many time steps and so may contain useful information about the far past, which should be opportunistically exploited.

Choosing an algorithm among BPTT and its truncated version essentially depends on both the length of the sequences we are dealing with, the average span of the temporal dependencies in our sequences, as well as the available computational capabilities (i.e. hardware resources). As TBPP can be seen as a reasonable compromise for training on considerably long sequences – which is inevitable for certain application domains, BPTT is always preferable if computationally affordable. A common practice is to start with mini-batch gradient descent and BPTT.

Nevertheless, while training a neural network with mini-batch gradient descent provides the modeller with a good trade-off between computational sustainability and quick convergence, in some contexts our cost function might be hard to traverse if the actual size of our batches is

small enough to resemble SGD (for this reason, MBGD is often called SGD). Moreover, using a constant learning rate for all the parameters of our model might be rather restricting during learning, and is generally associated to slower converge.

Therefore, as one of the objectives is to improve and accelerate the convergence of the learning process, we should consider to adopt one of the several extensions of the baseline gradient-based algorithms outlined before. According to the nature of the approaches, these accelerated optimisation methods can be grouped into the two following families: *constant learning rate* or a *per-parameter* optimisation algorithm, which are both described below.

Constant learning rate optimisation algorithms

As mentioned in the previous section, SGD (or mini-batch GD with small batch sizes) may have trouble navigating a cost function shaped as a long *shallow ravine* leading to the optimum. In this context, ravines identify areas where the surface curves much more steeply in one dimension than in another; the cost functions of deep architectures tend indeed to have this shape near local optima. In this scenario, SGD will tend to oscillate across the slopes of the ravine, since the negative gradient will point down one of the steep sides rather than along the ravine towards the optimum. Hence, as standard SGD makes hesitant progress along the bottom towards the local optimum, it would lead to slow convergence, particularly after the initial steep gains.

A more intuitive and metaphoric explanation of this problem can be obtained if we consider our cost function shaped like a long, narrow canyon that gradually slopes toward a minimum. Depending on the initialisation of our parameters, we start from some point on the canyon wall and the ultimate goal of the learning process is to reach such a minimum. However, as the canyon walls are much steeper than the gradual slope of the canyon toward the minimum, the negative gradient will point to the direction of steepest descent – mostly towards the canyon floor. This is where the problem occurs: if the size of our steps (i.e. the learning rate) is small enough, we could descend to the canyon floor, then follow it towards the minimum, but with slow progress/convergence. On the other hand, increasing the learning rate would be even worse because the direction of the steps would not be affected; as a consequence, we would overshoot the canyon floor and end up on the opposite wall. This behaviour is detrimental for the learning process because the same pattern would be repeated again and again: we oscillate from wall to wall while making slow progress toward the minimum, i.e. overshooting the gradient.

To address this common issue, the algorithms belonging to this family try to accumulate and exploit the information of the previous gradients in order to guarantee more consistent updates towards a local minimum. This is done while keeping the learning rate constant for all parameters. The two most common algorithms in this class are described as follows.

- **Momentum** is a simple method for pushing a quadratic cost function more quickly along

its shallow ravine (Qian, 1999). Intuitively, the algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.

More formally, this algorithm introduces a variable v, which is set to an exponentially decaying average of the gradients. This variable is also known as *velocity vector* because of its role during the new parameter update. In fact, v gives both the direction and speed at which the parameters move through parameter space (Goodfellow et al., 2016). The momentum term thus increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions (Ruder, 2016). As a result, we gain faster convergence and reduced oscillation of the cost function. The momentum update equations are given as follows:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \tag{3.43}$$

$$\theta = \theta - v_t \tag{3.44}$$

where a hyper-parameter $\gamma \in [0,1)$, called *momentum*, determines how quickly the contributions of previous gradients exponentially decay. The role of momentum is essential for convergence, as it damps the velocity to a certain (yet predefined) extent in order to avoid that our descent never comes to a stop at the bottom of the ravine.

More technically, γ is usually set to values such as $\{0.5, 0.9, 0.95, 0.99\}$, whereas a more preferable procedure would be that of using an annealing schedule (as similarly done for learning rates). In particular, as approaching the minimum of a cost function with a ravine form would benefit from different values of γ during the momentum descent, this hyper-parameter is generally increased in later stages of learning. A typical setting is to start with momentum of about 0.5 and anneal it to 0.99 or so over multiple epochs.

Finally, when using momentum, the learning rate η may need to be smaller since the magnitude of the gradient is expected to be larger in this settings.

- **Nesterov momentum** (Sutskever, Martens, Dahl, & Hinton, 2013) is a variation of momentum inspired by Nesterov's accelerated gradient method (Nesterov, 1983). It is said to exhibit stronger theoretical convergence guarantees for convex cost functions.

  The intuition behind Nesterov momentum is to implement a sort of look-ahead when computing the gradient, so as to anticipate the next parameter update. More precisely, when the current parameter vector is at some position θ in the ravine, then looking at the momentum update above and ignoring the second term with the gradient, we know that

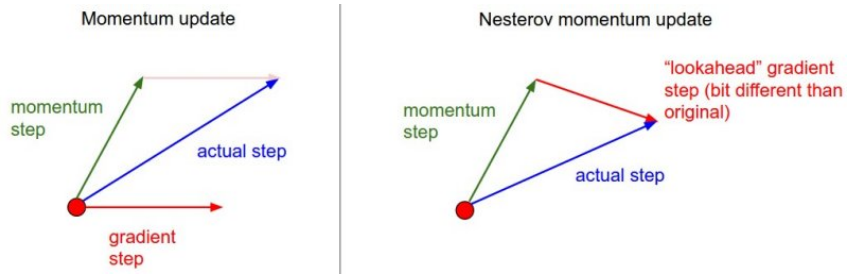


Figure 3.15: Standard momentum vs Nesterov momentum

the momentum term alone is about to nudge the parameter vector by γv. Therefore, if we are computing the gradient, we can exploit the future approximate position $\theta - \gamma v_{i-1}$ as a sort of look-ahead – a point in the vicinity of where we are actually going to end up soon. We can thus look ahead by calculating the gradient w.r.t. the approximate future position of our parameters as follows:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \tag{3.45}$$

$$\theta = \theta - v_t \tag{3.46}$$

As can be seen from the above equations, the difference between Nesterov momentum (Equation 3.45) and standard momentum (Equation 3.43) is where the gradient of the cost function is evaluated. In particular, Nesterov momentum evaluates the gradient after the current velocity vector is applied, as an attempt to add a correction factor to the standard method of momentum (Goodfellow et al., 2016). In other words, whereas standard momentum first computes the current gradient and then takes a big jump in the direction of the updated accumulated gradient, Nesterov momentum first makes a big jump in the direction of the previous accumulated gradient, measures the gradient and then makes a correction (Ruder, 2016). As a results, such an update prevents us from going too fast towards certain directions and increases responsiveness of the learning process.

Despite the fact that Nesterov momentum can dramatically improve the rate of convergence for batch gradient descent algorithms, it is worth remarking that such a desirable behaviour is not unfortunately observed in the stochastic gradient descent case.

Per-parameter optimisation algorithms

As strongly argued in the ML research community, the learning rate is one of the most difficult hyper-parameters to set/tune, as the impact on model performance is much more pronounced

than those of other "knobs". This is further exacerbated by the fact that the optimisation algorithms discussed so far manipulate the learning rate globally and equally for all parameters.

Since the cost function is often highly sensitive to some directions in the parameter space and insensitive to others, we can think of adapting our updates to each individual parameter, so as to perform larger or smaller updates depending on their contribution. This is the intuition behind the class of per-parameter optimisation algorithms. Although some of these methods still require setting other hyper-parameters, the argument is that they are well-behaved for a broader range of hyper-parameter values in comparison to the raw learning rate (Dozat, 2016).

The most common adaptive methods can be grouped into: L_2 *norm-based* algorithms and *combined* algorithms. The most popular algorithms of the first class are outlined below.

- **Adaptive Gradient** (Adagrad) (Duchi, Hazan, & Singer, 2011) is an L_2 *norm-based* optimisation algorithm adapting the learning rate to the parameters in such a way to perform larger updates for infrequent parameters and smaller updates for frequent ones. More formally, it divides the learning rate η at every step by the L_2 norm of all previous gradients; hence, we are slowing down learning along dimensions that have already changed significantly and speeding up learning along those that have changed slightly (Dozat, 2016).

  One clear advantage of Adagrad is the ability to stabilise the model's representations of common features in favour of those of rare features. For this reason, this algorithm is indicated for dealing with sparse data. The update equations are given as follows:

$$g_t = \nabla_{\theta_{t-1}} J(\theta_{t-1}) \tag{3.47}$$

$$n_t = n_{t-1} + g_t^2 \tag{3.48}$$

$$\theta_t = \theta_{t-1} - \eta \frac{g_t}{\sqrt{n_t} + \varepsilon} \tag{3.49}$$

  where g is just introduced for notational convenience, n_t is the vector accumulating the squared gradients for each parameter at update time t, and ε is a small non-negative constant (i.e. a smoothing term) to keep the denominator from being too small so as to avoid division by zero. This values is usually set in the range $[1e-4, 1e-8]$.

  As a result of its ability to adapt the learning rate to each parameter, according to the euclidean norm of their past gradients, Adagrad eliminates the need to manually tune the learning rate. In fact, most implementations use a default starting value of 0.01 and let the algorithm performs the per-parameter adaptation in Equation 3.49.

  However, when training deep neural networks, Adagrad may produce unstable configurations due to premature and excessive decrease in the effective learning rate. This happens

when the norm vector n eventually becomes so large that it causes the learning rate to shrink too much (eventually becoming infinitesimally small). As a direct consequence, training slows to a halt, thereby preventing the model from reaching any local minimum.

- **RMSProp** is another *L_2 norm-based* optimisation algorithm proposed by Tieleman and Hinton (2012) as an alternative to Adagrad. RMSProp was specifically designed to reduce the aggressive monotonically decreasing learning rate of the Adagrad update; this is easily achieved by replacing the sum in n_t with a decaying mean parameterised by a new hyper-parameter ν (i.e. the decay rate). This moving average of squared gradients allows the model to continue to learn indefinitely and it is defined as:

$$n_t = \nu n_{t-1} + (1-\nu) g_t^2 \tag{3.50}$$

  where the decay rate hyper-parameter ν is typically chosen among $\{0.9, 0.99, 0.999\}$. The parameter update equation was not reported as it is identical to Adagrad in Equation 3.49; the only difference lies in the gradient accumulator ν which is now a leaky variable. Hence, RMSProp still modulates the learning rate of each weight based on the magnitudes of its gradients, but unlike Adagrad, the updates do not get monotonically smaller.

  On an empirical point of view, RMSProp has been demonstrated to be an effective and practical optimisation algorithm for deep networks (Goodfellow et al., 2016), and is currently one of the go-to optimisation methods being employed routinely.

Whereas the constant learning rate optimisation algorithms (Section 3.3.1) are based on the intuition of moving the parameters more quickly along the dimensions whose gradients point in the same directions as previous ones, the so-far outlined per-parameter methods exploit the information of past gradients in order to perform larger updates for infrequent parameters and smaller updates for frequent ones. Therefore, another approach is to combine both these methods in such a way to leverage both their advantages in the resulting learning process. The two most common *combined algorithms* are outlined as follows.

- **Adaptive moment estimation** (Adam) (Kingma & Ba, 2014) combines the classical momentum with RMSProp to improve performance on a number of benchmarks.

  We can identify two main differences distinguishing Adam from a mere combination of the before mentioned optimisation algorithms: (i) in Adam, momentum is incorporated directly as an estimate of the first-order moment (using a decaying mean instead of a decaying sum) of the gradient; (ii) Adam includes bias corrections to the estimates of both the first-order moments and the second-order moments to compensate for the fact

that in the first few time steps these vectors are both initialised thus biased at zero. The moments update in Adam is defined by the following equations:

$$v_t = \mu v_{t-1} + (1-\mu) g_t \tag{3.51}$$

$$n_t = \nu n_{t-1} + (1-\nu) g_t^2 \tag{3.52}$$

where v_t and n_t are the estimates of the first-order moment (the mean) and the second-order moment (the uncentered variance) of the gradients respectively. Adding the bias correction mechanism before the "warm up" of the moment vectors thus results in the full Adam update schema, formulated as follows:

$$\hat{n}_t = \frac{n_t}{1-\beta_1^t} \tag{3.53}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{3.54}$$

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{v}_t}{\sqrt{\hat{n}_t} + \varepsilon} \tag{3.55}$$

where again, we are introducing two new hyper-parameters i.e. β_1 and β_2; in particular, Kingma and Ba (2014) recommend using default values of 0.9 for β_1, 0.999 for β_2, and 1e-8 for the learning rate η. The authors also demonstrated empirically that Adam works well in practice and compares favourably to other adaptive algorithms.

As we can now observe from the equations reported so far, RMSProp also incorporates an estimate of the second-order moment. However, as it lacks the correction factor of Adam, this estimate may have high bias early in training. For this reason, Adam is generally regarded as being fairly robust to the choice of hyper-parameters, though the learning rate sometimes needs to be changed from the suggested default (Goodfellow et al., 2016).

One last observation concerns the nature of the update of the second moment vector, which is peculiar to RMSProp. The n_t factor in the Adam update rule scales the gradient inversely proportionally to the L_2 norm of the past gradients (via the n_{t-1} term) and the current squared gradient g_t^2. In this context, we can think of generalising this update to the family of L_p norms, so as to choose the value of p that better suits our problem. This is indeed what Kingma and Ba (2014) proposed for the specific case of the infinity norm L_∞. The resulting algorithm – **AdaMax**, also removes the need for the bias-corrected $\hat{n}_t$. The update equations which differ from Adam are given as follows:

$$n_t = \max(\nu n_{t-1}, (1-\nu) g_t^2) \tag{3.56}$$

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{v}_t}{\sqrt{n_t} + \varepsilon} \tag{3.57}$$

We can thus apply the same consideration for all the L_2 norm-based algorithms. For instance, using the L_∞ norm in the denominator instead of the L_2 norm in the RMSProp update gives another adaptive procedure known as **MaxaProp**.

- **Nesterov Adam** (Nadam) (Dozat, 2016) is the natural extension of Adam as an attempt of combining RMSProp with Nesterov momentum. Therefore, in order to incorporate Nesterov momentum into Adam, an intuition proposed by the author of this algorithm is considered which modifies its momentum term v_t accordingly. To define this algorithm, we first recall the Nesterov momentum update rule as follows:

$$\begin{aligned} g_t &= \nabla_{\theta_{t-1}} J(\theta_{t-1} - \gamma v_{t-1}) \\ v_t &= \gamma v_{t-1} + \eta g_t \\ \theta_t &= \theta_{t-1} - v_t \end{aligned} \tag{3.58}$$

Nevertheless, instead of applying the momentum step twice (once for updating the gradient g_t and a second time for updating the parameters), Dozat (2016) proposed to apply the look-ahead momentum vector directly to update the current parameters.

$$\begin{aligned} g_t &= \nabla_{\theta_{t-1}} J(\theta_{t-1}) \\ \bar{v}_t &= \gamma v_t + \eta g_t \\ \theta_t &= \theta_{t-1} - \bar{v}_t \end{aligned} \tag{3.59}$$

As noticeable, rather than using the momentum vector v_t for the update, we now exploit the current momentum vector $\bar{v}_t$ as a look-ahead. Therefore, integrating Nesterov momentum in Adam requires solely the replacement of our momentum vector. This leads to the following update rules (for convenience, we only report those differing from Adam):

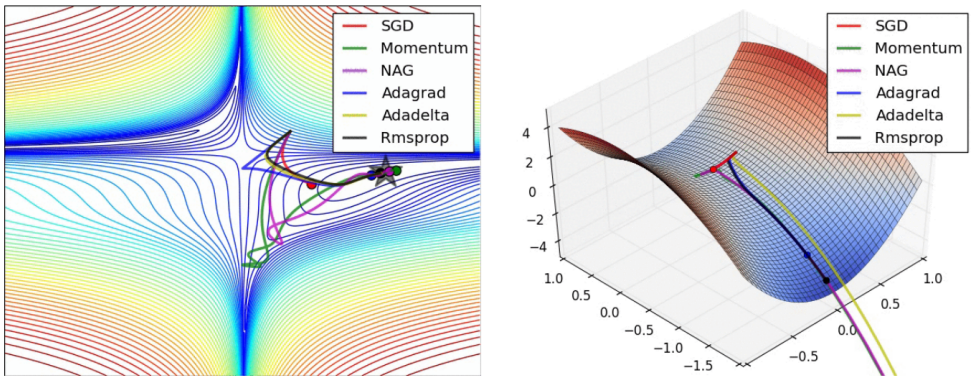


Figure 3.16: **Left**: Contours of a Beale cost function and time evolution of different optimisation algorithms, to highlight the overshooting behaviour of constant learning rate algorithms. **Right**: Simplified visualisation of a saddle point, i.e. a point where one dimension has a positive slope, while the other has a negative slope; this surface poses issues to pure SGD because of its symmetric behaviour. All credits to Alec Radford.

$$\bar{v}_t = \mu \hat{v}_t + (1-\mu) g_t \tag{3.60}$$

$$\theta_t = \theta_{t-1} - \eta \frac{\bar{v}_t}{\sqrt{\hat{n}_t} + \varepsilon} \tag{3.61}$$

As Nesterov momentum is theoretically and often empirically superior to momentum, it seems reasonable to prefer Nadam over Adam as a general suggestion. To support this consideration, Dozat (2016) tested Nadam on three benchmark tasks (i.e. handwritten character classification, word2vec, language modelling), in order to compare its performance with that of the other optimisation algorithms outlined in this section.

This experiment demonstrated a consistent improvement of Nadam over Adam in term of test error and quicker convergence, and more generally over the other adaptive learning algorithms. The author concluded that *"if one is going to use momentum with RMSProp to train their model, they may as well use Nesterov momentum"* (Dozat, 2016).

Nevertheless, it is worth remarking that the same consideration does not hold for recurrent neural networks, as the results for the benchmark of language modelling indicated a clear and marked preference over traditional (non-adaptive) momentum algorithms.

Choosing an optimisation algorithm

Choosing the right optimisation algorithm for a problem is again a process of trial and error, as there does not exist neither theoretical nor empirical evidence of a single optimisation algorithm, or a family of them, outperforming the others in every context. In sum, the choice of optimisation algorithm can be considered as an additional hyper-parameter.

In practice, empirical results suggested that the family of algorithms with adaptive learning rates tend to perform robustly (Goodfellow et al., 2016). This argument seems to acquire more relevance when dealing with certain types of data and when the learning task is complex enough to obstruct the modeller to make possible guesses and assumptions on the cost function. Overall, the main advantages of adaptive optimisation algorithms are summarised below.

- If dealing with *sparse data*, the stabilisation-oriented behaviour of Adagrad (and its extensions) can be desirable: performing larger updates for infrequent parameters and smaller updates for frequent ones, is indeed more likely to improve convergence in this context.

- Cost functions with *particular shapes* or with *saddle points* can be hard to optimise with standard constant learning rate algorithms. For instance, as illustrated in Figure 3.16, when the cost function resembles the shape of a Beale function, per-parameter optimisation algorithms tend to immediately head off in the right direction and converge rather fast; on the other hand, momentum and Nesterov momentum were led off-track (we can think of this process as balls rolling down the hill).

  Similarly, saddle points are challenging for constant learning rate algorithms because their symmetric behaviour in the parameters' update makes it difficult to escape from oscillating in the vicinity of that point. Nevertheless, momentum and Nesterov momentum eventually manage to escape the saddle point. Again, adaptive algorithms manage to head down the negative slope with no particular delay.

- Finally, an additional benefit that we cannot neglect is the reduced dependence from *tuning the learning rate*. Considering the significant influence of this hyper-parameter on constant learning rate algorithms, a careful procedure of learning rate tuning would be required to achieve a good convergence. On the other hand, adaptive methods are more robust to the initialisation of the learning rate, because of their ability to dynamically adapt it for the needs of each parameter in the model.

Therefore, the following question arises: why should we use a constant learning rate optimisation algorithms if the adaptive ones are so powerful? As far as we know, there are some specific contexts where the family of per-parameter algorithms do not deliver convincing performances. For instance, in the experiments performed by Dozat (2016), using adaptive

methods on the task of language modelling was found to hinder performance even after the hyper-parameters were extensively tuned. The author also added that it is still not clear why it is conceivable that the assumptions adaptive methods make about how features are represented do not hold in the case of language models, or RNNs in general (Dozat, 2016).

This might be one of the reasons why Ruder (2016) remarks that "*many recent papers still use vanilla SGD without momentum and a simple learning rate annealing schedule*". To conclude, the choice of the optimisation algorithm seems to depend on several factors. Thus, the modeller's familiarity and experience with optimisation algorithms, with special concern on the hyper-parameters and their effect, often tips the balance of this decision.

Batch normalisation: an optimisation template with regularisation effects

Besides the optimisation algorithms outlined so far, we can also consider an additional technique that is not an optimisation algorithm per se, but can be used as a general template to incorporate into the training process. This optimisation template – specifically designed for deep neural network, is called *batch normalisation*. The main idea is to reparameterise the model to introduce both additive and multiplicative noise on the hidden units at training time. Although the primary goal is to improve optimisation by means of an adaptive reparametrisation, the noise introduced can also have a regularising effect. The problem batch normalisation intends to address is known as *Internal Covariate Shift*, which is briefly explained as follows.

Deep networks involve the composition of several layers, and the gradient tells how to update each parameter on each layer under the assumption that the other layers (below) remain unchanged. However, this is a drastic simplification considering that layers are updated simultaneously, hence unexpected results can happen after the update. More precisely, one can think of an arbitrary layer as receiving samples from a distribution that is shaped by the layer immediately below. Because of the way gradient descent works, this distribution changes during the course of training, making any layer but the first responsible not only for learning a good representation but also for adapting to a changing input distribution (Laurent, Pereyra, Brakel, Zhang, & Bengio, 2016). This distribution variation is termed *Internal Covariate Shift*, and it degrades the efficiency of training as it distracts "the work of each hidden unit".

Batch normalisation (Ioffe & Szegedy, 2015) addresses the problem of coordinating updates across layers; it works by standardising the activations of a layer using empirical estimates of their means and standard deviations (Ba, Kiros, & Hinton, 2016). More formally, we denote with H a mini-batch of activations of the layer we want to normalise, with the activations for each sample appearing in a row of H. To normalise these activations, we perform this update:

$$H' = \frac{H - \mu}{\sigma} \tag{3.62}$$

where μ and σ are the vectors containing the mean and the standard deviation of each unit in the mini-batch respectively. In other words, we are just broadcasting the vectors μ and σ to be applied to every row of the mini-batch activation matrix H. Assuming to deal with a mini-batch of m training sample, at training time we compute:

$$\mu = \frac{1}{m}\sum_i H_{i,:} \tag{3.63}$$

$$\sigma = \sqrt{\delta + \frac{1}{m}\sum_i (H-\mu)_i^2} \tag{3.64}$$

where δ is a small positive constant (e.g. 10^{-8}) to improve numerical stability. The rest of the network then operates on H' as if nothing had happened. Since the normalisation is now part of the network, the back propagation procedure needs to be adapted to propagate gradients through the mean and variance computations as well (Laurent et al., 2016). The normalisation operations thus ensures that the gradient will never propose an operation that acts simply to increase the standard deviation or mean of each element in H' (Goodfellow et al., 2016).

However, standardising the intermediate activations indirectly reduces the representational power of the layer. To address this, batch normalisation also introduces two learnable parameters γ and β which respectively scale and shift H' to have any mean and standard deviation. The resulting parametrisation can represent the same family of functions of the input as the old parametrisation, but the mean is now solely determined by β and it is even easier to learn with gradient descent. The new layer is thus defined as follows:

$$BN(H) = \gamma \odot H' + \beta \tag{3.65}$$

At test time, we cannot use the statistics of the mini-batch to perform regularisation, because we need to process one test sample at a time. Alternatively, we can use an estimate by either forwarding and averaging the statistics of several training mini-batches, or by maintaining a running average calculated over each mini-batch seen during training (Laurent et al., 2016).

Nonetheless, despite its simplicity and the benefits of significant training speed-ups and more generalisation capabilities when implemented in feed-forward networks, batch normalisation applied to recurrent neural network was proven to be difficult. More precisely, Laurent et al. (2016) pointed out that when batch normalisation was applied to RNNs, as proposed in (Ioffe & Szegedy, 2015), it did not help the training procedure in their benchmark experiments.

To address this obstacle, with the ultimate goal of bringing the benefits of batch normalisation to the realm of recurrent neural networks, several studies attempted to revisit the original

optimisation template proposed by Ioffe and Szegedy (2015). To the best of our knowledge, the most relevant works in this direction can be outlined as follows.

- Laurent et al. (2016) hypothesised that applying vanilla batch normalisation affects training negatively because the repeated rescaling applied over time amplifies the risk of exploding gradients for long sequences. Therefore, they propose to to apply batch normalisation only to the input-to-hidden transition. In other words, batch normalisation is applied only on the vertical connections (i.e. from one layer to another) and not on the horizontal connections (i.e. within the recurrent layer).

  As outlined in Section 3.2.3, this idea is similar to the way Zaremba et al. (2014) applied dropout to recurrent neural networks. More generally, the impression is that whenever a method does work on feed-forward NNs but not on RNNs, one should isolate the application of that method only on the non-recurrent connections of the network.

- Cooijmans, Ballas, Laurent, Gülçehre, and Courville (2016) run counter to the previous hypothesis as they show that the difficulties with recurrent batch normalisation reported in the previous work are due to improper initialisation of the batch normalisation parameters, and γ in particular. More precisely, the authors demonstrate that it is both possible and highly beneficial to apply batch normalisation (as in (Ioffe & Szegedy, 2015)) also in the hidden-to-hidden transition of recurrent models. Therefore, during training we need to estimate the statistics across the mini-batch independently for each time step. This translate into storing separate vectors of mean and standard deviation for each time step.

- Ba et al. (2016) observed that, as the summed inputs to the recurrent neurons in an RNN often vary with the length of the sequence, storing different statistics for different time-steps can be inconvenient for sequences of different lengths and sometimes impractical for long ones. Furthermore, batch normalisation cannot be applied to online learning tasks or to extremely large distributed models where the mini-batches have to be small.

  Driven by these considerations, the authors introduced *layer normalisation*, which directly estimates the normalisation statistics from the summed inputs to the neurons within a hidden layer instead of requiring mini-batches of data. In this way, layer normalisation does not introduce any new dependencies between training cases, and the resulting approach is possibly applicable to any ANN.

3.3.2 Hyper-parameter selection

As observed throughout this section, choosing the optimal hyper-parameters of our model and for the training procedure is crucial. Some of these hyper-parameters affect the temporal and

memory complexity of running the training routines, whereas others control the generalisation capabilities of the trained model – its ability to infer correct results on new (unseen) inputs.

Overall, hyper-parameters can be organised into the following categories: *architectural hyper-parameters* and *training-based hyper-parameters*. Examples of the first category include the number of layers, the type of activation function for each layer, the number of hidden units per layer, the choice of a baseline model for a sequence learning task as well as the configuration of the recurrent connections. On the other hand, training-based hyper-parameters include the learning rate, the regularisation strength (e.g. weight decay penalty, dropout strength), the loss function and the several parameters for our adaptive optimisation algorithms.

Choosing the optimal hyper-parameters for virtually any machine learning model seems to be more of an art than a science. Nonetheless, we can identify two basic approaches to choosing these hyper-parameters: *choosing them manually*, which requires understanding what they do and how machine learning models achieve good generalisation; and *choosing them automatically*, which reduces the need to understand these ideas, but also comes with increased computational costs. Both these approaches are outlined in the following sections.

Manual hyper-parameter tuning

Setting the hyper-parameters manually requires both familiarity with the type of models we are dealing with, as well as considerable experience of training methods at hand. The modeller is thus expected to understand the relationships between hyper-parameters, training error, generalisation error and computational resources (memory and runtime) (Goodfellow et al., 2016).

To achieve a reasonable balance between learning objectives and the available computational resources, the goal of manual hyper-parameter search is to find the lowest generalisation error subject to some runtime and memory budget. The generalisation error can be defined as the average value of the loss function evaluated on the test set; thus it provides a concrete measure (assuming that the test set is representative enough) of the model's ability to generalise to novel, previously unseen observations/contexts.

Assuming that our learning task can be addressed with the type of models we are dealing with, and that we have enough training data, the generalisation error typically follows a U-shaped curve when plotted as a function of one of the hyper-parameters: at one extreme, the hyper-parameter value corresponds to low capacity, and generalisation error is high because training error is high (the underfitting regime); on the opposite, the hyper-parameter value corresponds to high capacity, and the generalisation error is high due to the gap between training and test error (the overfitting regime). Therefore, our objective is to find a proper configuration in the middle achieving optimal model capacity – the lowest possible generalisation error.

In light of these considerations, careful and attentive monitoring of the generalisation error

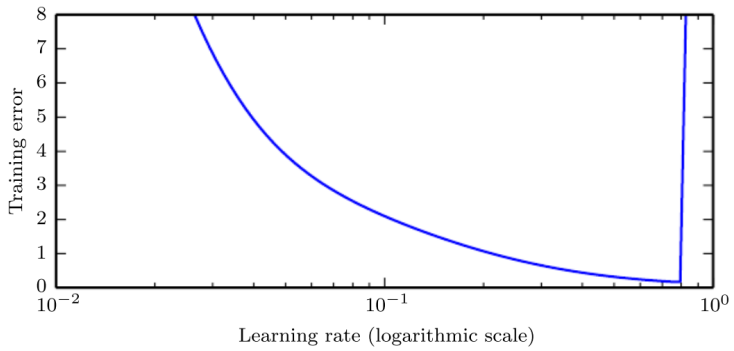


Figure 3.17: Relationship between the learning rate and the training error for a fixed training time. Generalisation error can follow this curve or be complicated by regularisation effects arising out of having a too large or too small learning rates. Source: (Goodfellow et al., 2016)

.

as the learning process goes on would allow us to adjust the effective capacity of the model to match the complexity of the task. In other words, we need a model that is perfectly tailored to our learning task. As argued in (Goodfellow et al., 2016), effective capacity is constrained by three factors: (i) the representational capacity of the model; (ii) the ability of the learning algorithm to successfully minimise the cost function used to train the model; (iii) and the degree to which the cost function and training procedure regularise the model. Therefore, as manual hyper-parameter tuning is driven by experience and familiarity, we can identify a few heuristics addressed to each of these limitations of a model's effective capacity.

- **Number of hidden layers and units**. In case the training error is much higher than our target error, we need to increase the representational capacity of our model. In particular, if we are still not using regularisation and we are also confident that the optimisation algorithm is performing as we expect, then we may consider adding more hidden layers or add more hidden units per layer. The resulting model would have higher representational capacity, i.e. it is capable of representing more complex functions.

- **Learning rate**. The learning rate is probably the most important hyper-parameter, as it controls the effective capacity of the model in a more complex way than other hyper-parameters: the effective capacity of the model is highest when the learning rate is correct for the optimisation problem, not when the learning rate is especially large or especially small (Goodfellow et al., 2016). More precisely, as we can see in Figure 3.17, the learning rate exhibits a U-shaped curve for the training error: when too large, we would overshoot the gradient and jeopardise the convergence of the optimisation algorithm; when too small, training is not only slower, but may become stuck with a high training error.

- **Regularisation**. In case the test error is much higher than the target error rate we aim to

achieve, we need to improve the generalisation capabilities of our model by reducing the effective model capacity. As discussed in Section 3.2.3, we may consider to increase the coefficients of the adopted regularisation strategy(ies), if any. In fact, the common belief is that the best performance comes from a large model that is regularised well.

Nevertheless, we can use the opposite approach in case we end up with a potentially underfitted model. In that case, the contribution of the chosen regularisation strategies might be too strong, and we should at least try to reduce their influence. For instance, decreasing the weight decay coefficient would free the model parameters to become larger decreased. Similarly, dropping units less often would allow them to collaborate more with each other in order to focus on the problem and fit the training set.

- **Using more data**. If we can manage to keep a low training error, it is ideally possible to further reduce the generalisation error by collecting more training data. This can be easily done by continually increasing our model capacity and training set size until the task is solved. This approach does of course increase the computational cost of training and inference, so it is only feasible given appropriate resources.

Apart from any general guidelines, it is worth remarking that the effectiveness of manual hyper-parameter tuning strictly depends on the experience of the user in exploring hyper-parameter values for neural networks applied to similar tasks. For this reason, it is often preferred to start with an automated approach in order to accumulate experience in the long run.

Automated hyper-parameter tuning

These strategies can be useful to understand how our neural network is affected by the choice of the hyper-parameters, and to parallelise the computational burden of training multiple models (with different hyper-parameters) on multiple processing units. The most common automated hyper-parameter tuning procedures are outlined in the following list.

- **Model-based search** is driven by the intuition that the process of finding good values for our hyper-parameters can be cast to the following optimisation problem: hyper-parameters play the role of decision variables; the objective to minimise is the validation error that results from training using such hyper-parameters; and the constraints of that problem might express the available budget for training time and memory.

  In rare learning contexts, computing the gradient of a differentiable error measure on the validation set w.r.t. the hyper-parameters, might be feasible and convenient as it would allow us to use gradient descent. However, in most practical cases computing the gradient

is not possible, either due to the high complexity of this operation, or to hyper-parameters having intrinsically non-differentiable interactions with the validation set error.

As an alternative, we can define a model of the validation set error and perform optimisation to iteratively generate and evaluate new hyper-parameter guesses. Most model-based algorithms for this task are based on a Bayesian regression model to estimate both the expected value of the validation set error for each hyper-parameter, and the uncertainty around this expectation. The resulting optimisation strategy permits to perform both:

1. *exploration* in the hyper-parameter space, i.e. proposing hyper-parameters for which there is high uncertainty, which may lead to a large improvement (but may also perform poorly in case we end up in a region with high validation error);
2. and *exploitation*, i.e. proposing similar hyper-parameters which the model is confident will perform as well as any hyper-parameters it has seen so far.

According to Goodfellow et al. (2016), Bayesian approaches to hyper-parameter optimisation tend to perform comparably to human experts, but can also fail on some problems.

- **Grid search** is one of the most intuitive procedures for tuning the hyper-parameters of our model, yet it is affordable if we are tuning a few of them. The idea consists in defining a (small) set of possible values to explore for each hyper-parameter so that we can train as many models as the number of combinations among them – the Cartesian product of the set of values for each individual hyper-parameter. After all model configurations are trained, the model instance with the best validation set error is then selected for testing.

  By doing so, the modeller is only expected to define the set of candidate values for each hyper-parameter, depending on their type/domain. For instance, a grid search involves picking values on a logarithmic scale, e.g. a learning rate taken within the set {1e-1, 1e-2, 1e-3, 1e-4}, or a number of hidden units taken with the set $\{100, 200, 500, 700\}$.

  As such, grid search assumes that the modeller already has some experience with tuning hyper-parameters. Moreover, it is also crucial to be aware of the computational requirements of such procedure. By definition, the computational cost of grid search grows exponentially with the number of hyper-parameters: if we are tuning m hyperparameters, each of which takes at most n values, then the number of models to train independently grows as $O(n^m)$. For example, in the very optimistic scenario in which we need to optimise 5 parameters, each with 10 possible values, then we would have to train and evaluate 10^5 models. Assuming that our network can train in 10 minutes, we would finish tuning our hyper-parameters in almost 2 years on a single device.

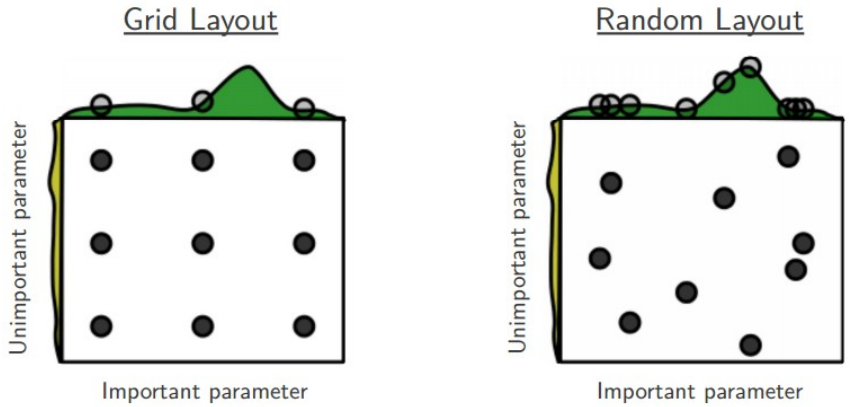


Figure 3.18: Grid search vs random search. In this figure we can observe that only the hyper-parameter on the horizontal axis has significant influence on the validation error. Whereas grid search would waste an amount of computation that is exponential in the number of non-influential hyper-parameters, random search tests a unique value of every influential hyper-parameter on nearly every trial. Figure taken from (Bergstra & Bengio, 2012).

.

Hyper-parameter name	**Possible distribution**	**Possible values**
Learning rate	Log-Uniform	$x \in [0.0001, 0.1]$
Mini-batch size	Log-Uniform integer	$x \in [16, 1024]$
Activation function	Multinoulli	$x \in \{\text{ReLU, PReLU, sigmoid, tanh}\}$
Number of hidden layer	Multinoulli	$x \in \{1, 2, 3\}$
Number of units in the first hidden layer	Uniform integer	$x \in [50, 200, 500]$
Dropout probability	Uniform	$x \in [0, 0.5]$

Table 3.1: Examples of hyper-parameter distributions for random search.

- **Random search** is similar to grid search, but instead of trying all possible combinations of hyper-parameters, it only uses a randomly selected subset of them. The resulting procedure is relatively simple to program, more convenient to use, and converges much faster to good values of the hyper-parameters as demonstrated in (Bergstra & Bengio, 2012).

  To perform random search we need to define a marginal distribution for each hyper-parameter, e.g. a Bernoulli or Multinoulli for binary or discrete parameters, or a uniform distribution on a log-scale for positive real-valued hyper-parameters (Goodfellow et al., 2016). As we can often assume that hyper-parameters are independent from each other, the search algorithm then randomly samples from the resulting joint distribution of hyper-parameter configurations and runs training with each of them.

  One of the major benefits of random search, in contrast to grid search, is that we do not need to discretise or bin the candidate values of our hyper-parameters. Hence, even a modeller that is still not familiar with the hyper-parameter space defined by their problem can explore a larger set of values at a reasonable computational cost. This is increasingly

more evident when we are dealing with several hyper-parameters that do not considerably affect the validation error (a simple example is given in Figure 3.18). Indeed, as observed by Bergstra and Bengio (2012), random search reduces the validation set error much faster than grid search, in terms of the number of trials run by each method.

In practice, the whole process of hyper-parameter selection is rarely carried out by means of a single strategy; it is rather a combination of random and grid search together with manual tuning at multiple stages. It is common, indeed, that modellers try to leverage their experience to further narrow down the space of candidate hyper-parameters, so that the random sampling behaviour can be more effective in finding optimal configurations. Analogously, if the results accumulated by any empirical search suggest a particular direction/trend, modellers can intervene a posteriori to re-define the marginal distributions of each hyper-parameter.

3.4 Analysis and evaluation of RNNs

As introduced in the previous chapter, it is good practice to provide a formal and empirical evaluation when introducing novel neural network architectures, especially if the model can be used for a variety of tasks. The former evaluation can be achieved with architectural complexity measures (Section 3.4.1), providing a formal analysis of the structural properties of an ANN.

Since recurrent neural networks are mainly designed for sequential data, it is also important to agree on a common testbed of sequence modelling tasks on which RNNs can be evaluated. As we will see in Section 3.4.2, the research community is currently adopting a mixture of synthetic and real benchmark tasks, which can be helpful to provide an empirical measure regarding how well a recurrent neural network can perform in capturing long-term dependencies.

3.4.1 Architectural complexity measures

The availability of a theoretical framework for the formal analysis of recurrent architectures is a valuable resource that would allow modellers to: (i) measure architectural properties of RNNs in a formal way; (ii) trace any theoretical motivations behind the learning capabilities of the proposed sequence model. Compared to feed-forward architectures, recurrent connections pose a number of technical challenges for any analytical framework addressing RNNs. In fact, recurrent neural networks undergo multiple transformations – not only feed-forwardly (from input to output within a time step) but also recurrently (across multiple time steps).

Driven by these motivations, S. Zhang et al. (2016) introduced a general formulation of the connecting architectures of RNNs and defined three architectural complexity measures, i.e. *recurrent depth*, *feed-forward depth*, and *recurrent skip coefficient*. To outline the definitions

of both, this section provides a summary of the most relevant concepts from (S. Zhang et al., 2016), which are needed to explain these measures.

To formalise the concept of the *connecting architecture* we first need to extend the traditional graph-based illustration of RNNs to a more general definition with a *finite directed multigraph* and its unfolded version. Therefore, we first present the notion of the RNN cyclic graph $\mathcal{G}_c$ where the weights of edges are used to represent time delay differences between the source and destination node in the unfolded graph; this is analogous to the concepts of *skip connections through time* and *removing connections* outlined in Section 3.1.2.

Definition 3.4.1. Let $\mathcal{G}_c = (V_c, E_c)$ be a weighted directed multigraph, i.e. a directed graph allowing multiple directed edges connecting the same couple of nodes, where $V_c = V_{in} \cup V_{out} \cup V_{hid}$ is a finite non-empty set of nodes, $E_c \subset V_c \times V_c \times \mathbb{Z}$ is a finite set of directed edges. Each $e = (u, v, \sigma) \in E_c$ denotes a directed weighted edge pointing from node u to node v with an integer weight σ representing the delay of that connection. Each node $v \in V_c$ is labelled by an integer tuple (i, p) s.t. $i \in \{0, 2, \ldots, m-1\}$ denotes the time index of the given node, where m is the period number of the RNN, and $p \in S$ where S is a finite set of node labels.
Therefore, $\mathcal{G}_c = (V_c, E_c)$ is an RNN cyclic graph if: (i) for every edge $e = (u, v, \sigma) \in E_c$, let i_u and i_v denote the time index of the corresponding nodes, then $\sigma = i_v - i_u + k \cdot m$ for some $k \in \mathbb{Z}$; (ii) there exists at least one directed cycle, i.e. a closed walk with no repetitions of edges, in $\mathcal{G}_c$; (iii) for any closed walk ω, the sum of all the σ along ω is not zero. Here, the period number m is defined as the time length of the non-repeated recurrent structure in its unfolded RNN graph[3].

The last three conditions formally characterise a *folded* RNN: the first guarantees that a recurrent network unfolds consistently with its recursive pattern; the second requires the presence of at least one recurrent connection – ruling out feed-forward networks from this definition; the last condition avoids further cycles after unfolding. In light of these definitions, given an RNN cycle graph, the unfolding procedure of $\mathcal{G}_c$ over time $t \in \mathbb{Z}$ is formalised as follows.

Definition 3.4.2 (Unfolding). Given an RNN cycle graph $\mathcal{G}_c = (V_c, E_c)$, we define a new infinite set of nodes $V_{un} = \{(i + km, p) | (i, p) \in V_c, k \in \mathbb{Z}\}$. The new set of edges $E_{un} \in V_{un} \times V_{un}$ is constructed as $((t, p), (t', p')) \in E_{un}$ if and only if there is an edge $e = ((i, p), (i', p'), \sigma) \in E_c$ s.t. $t' - t = \sigma$ and $t \equiv i (\mathrm{mod}\ m)$. The new directed graph $\mathcal{G}_{un} = (V_{un}, E_{un})$ is called the unfolding of $\mathcal{G}_c$ and any infinite directed graph derived through unfolding is an RNN unfolded graph.

The RNN reported in Figure 3.19(a) is different from traditional RNNs using simple recurrent connections and following an unitary pattern when unrolled ($m = 1$). Conversely, we can find both skip connections and removed connections (introduced in Section 3.1.2. For instance,

[3]Most traditional RNNs have unitary period number ($m = 1$).

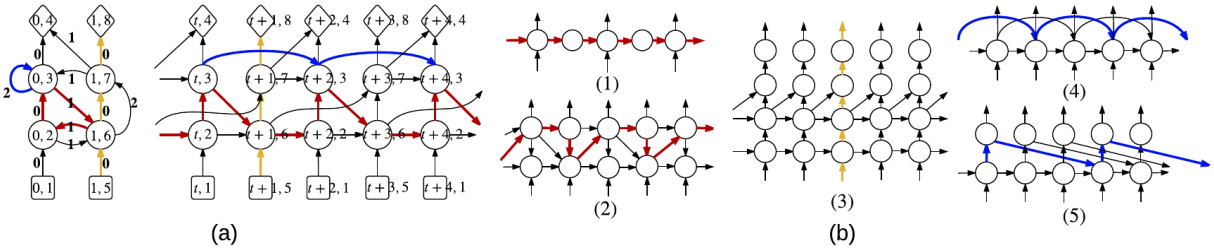


Figure 3.19: (a) An example of folded and unfolded RNN. To simplify the illustration, input nodes (in the set V_{in}) are denoted by squares, hidden nodes (in V_{hid}) are denoted by circles, and output nodes (in V_{out} are represented by diamonds. As we can observe, in the folded graph on the left, the number on each edge indicates the delay of this connection σ. The longest path is coloured in red. The longest input-output path is coloured in yellow and the shortest path is coloured blue. (b) other examples of complex unfolded RNNs.

the node $(0,3)$ is not directly connected to node $(1,7)$ which operates at the next time step, but a delayed connection exists between $(0,3)$ and $(2,11)$; this is an example of removed connection. Therefore, the connecting architecture formalised in the previous definitions describes how information flows among RNN units, and it is general enough to define any recurrent networks, independently of the implementation of the recurrent units.

Definition 3.4.3 (RNN). An RNN is a tuple $(\mathcal{G}_c, \mathcal{G}_{un}, \{F_{\hat{v}}\}_{\hat{v}\in V_c})$ in which $\mathcal{G}_{un} = (V_{un}, E_{un})$ is the unfolding of RNN cyclic graph $\mathcal{G}_c$, and $\{F_{\hat{v}}\}_{\hat{v}\in V_c})$ is the set of transition functions. In the forward pass, for each hidden and output node $v \in V_{un}$, the transition function $F_{\hat{v}}$ takes all incoming nodes of $\hat{v}$ as the input to compute the output. A recurrent neural network is said to be *homogeneous* if all the hidden nodes share the same form of transition function.

Now that we showed how an RNN can be formally described, and how different architectures may arise from such a flexible and generalised characterisation, we can proceed with the architectural measures. All these measures assumes homogeneous and unidirectional RNNs, although alternative definitions can be easily obtained by casting eventual bi-directional or non-homogeneous recurrent networks (S. Zhang et al., 2016).

- **Recurrent depth**. As the conventional definition of *depth* in ANNs matches the maximum number of nonlinear transformations from inputs to outputs, adapting this definition to RNNs should take into account the recurrent perspective spanning across time steps.

  For this purpose, additional notation is needed. Given an unfolded RNN graph $\mathcal{G}_{un}$, $\forall i, n \in \mathbb{Z}$, we denote with $\mathcal{D}_i(n)$ the length of the longest path from any node at starting time i to any node at time $i+n$. Hence, from the recurrent perspective, it is interesting to analyse how $\mathcal{D}_i(n)$ changes over time. In other words, as $\mathcal{D}_i(n)$ increases as n increases for all i, we aim at observing the rate of change of this value as n gets larger with sequences.

More formally, since $\mathcal{D}_i(n)$ approaches ∞ as n approaches ∞ as well, to measure the complexity of $\mathcal{D}_i(n)$ the authors consider its asymptotic behaviour, i.e. the limit of $\frac{\mathcal{D}_i(n)}{n}$ as $n \to \infty$. Under mild assumptions, this limit exists and the following theorem prove its computability and well-definedess (as thoroughly showed in (S. Zhang et al., 2016)).

Theorem 3.4.1. *Given an RNN and its two graph representations $\mathcal{G}_c$ and $\mathcal{G}_{un}$, we denote $C(\mathcal{G}_c)$ to be the set of directed cycles in $\mathcal{G}_c$. For $\vartheta \in C(\mathcal{G}_c)$, let $l(\vartheta)$ denote the length of ϑ and $\sigma_s(\vartheta)$ denote the sum of edge weights σ along ϑ. Under mild assumptions:*

$$d_r = \lim_{n\to\infty} \frac{\mathcal{D}_i(n)}{n} = \max_{\vartheta \in C(\mathcal{G}_c)} \frac{l(\vartheta)}{\sigma_s(\vartheta)} \tag{3.66}$$

In the above equation, d_r is defined as the recurrent depth of the RNN. Intuitively, we can think of this value as a measure of the average number of nonlinear transformation per time step as n gets large. For instance, if we consider the RNN reported in Figure 3.19(a), one can easily verify that, because of the complex recurrent architecture, $\mathcal{D}_t(1) = 5$, $\mathcal{D}_t(2) = 6$, $\mathcal{D}_t(3) = 8$ and so forth; therefore, the mathematical sequence $\frac{\mathcal{D}_t(1)}{1} = 5$, $\frac{\mathcal{D}_t(2)}{2} = 3$, $\frac{\mathcal{D}_t(3)}{3} = \frac{8}{3}$, ... eventually converges to $\frac{3}{2}$ as n tends to infinity. As we can observe, the recurrent depth d_r coincides with the number of nodes to traversed by the red path per time step; this suggests that it is possible to compute d_r from $\mathcal{G}_{un}$.

To conclude, it is worth considering that those recurrent architectures known as *deep RNNs*, resulting from stacking multiple recurrent layers on the top of each other, share the same recurrent depth of traditional shallow RNNs, which is equal to 1. Thus, we should not be surprised that our really deep RNN is indeed comparable to a shallow one if analysed on a recurrent depth perspective. This fact is often neglected, since one would typically consider the number of layers as a measure of depth, and think of stacked RNNs as "deep" and simple RNNs as "shallow" (S. Zhang et al., 2016).

- **Feed-forward depth**. As the recurrent depth only captures the number of nonlinear transformations in the time direction, we also need a measure in the feed-forward direction.

  To put more emphasis on the specific paths connecting inputs to outputs, let $\mathcal{D}_i^*(n)$ denote the length of the longest path from any input node at time step i to any output node at time step $i+n$ of an unfolded RNN graph $\mathcal{G}_{un}$. For small values of n, the recurrent depth d_r is not a good approximation for $\mathcal{D}_i^*(n)$ as it strongly depends on the feedforward depth (denoted as d_f). The following proposition guarantees the existence of such a quantity and demonstrates the role of both measures in quantifying the non-linearity of an RNN.

  Proposition 3.4.1. *Given an RNN with recurrent depth d_r, the supremum feed-forward*

depth, denoted as $d_f = \sup_{i,n\in\mathbb{Z}} \mathcal{D}_i^*(n) - n \cdot d_r$*, exists with an upper bound for* $\mathcal{D}_i^*(n)$*:*

$$\mathcal{D}_i^*(n) \leq n \cdot d_r + d_f \tag{3.67}$$

This proposition is thus helpful to understand the relationships between feed-forward and recurrent depths as the RNN is unrolled for several time steps. In particular, the above upper bound can be interpreted as follows: when n is small, $\mathcal{D}_i^*(n)$ is largely bounded by d_f; and when n is large the ball passes to to d_r which controls the nature of the bound. Thus, these measure are both important as they can capture the maximum number of nonlinear transformation of an RNN in the long and in the short run.

Theorem 3.4.2. *Given an RNN and its two graph representations* $\mathcal{G}_{un}$*,* $\mathcal{G}_c$*, we denote* $\xi(\mathcal{G}_c)$ *the set of directed paths that start at an input node and end at an output node in* $\mathcal{G}_c$*. For* $\gamma \in \xi(\mathcal{G}_c)$*, denoting* $l(\gamma)$ *the length and* $\sigma_s(\gamma)$ *the sum of* σ *along* γ *we have:*

$$d_f = \sup_{i,n\in\mathbb{Z}} \mathcal{D}_i^*(n) - n \cdot d_r = \max_{\gamma\in\xi(\mathcal{G}_c)} l(\gamma) - \sigma_s(\gamma) \cdot d_r \tag{3.68}$$

where m is the period number of the recurrent neural network defined by $\mathcal{G}_c$*.*

Again, in order to have a practical idea of the feed-forward depth, we can observe that for the RNN reported in Figure 3.19(a) this measure is equal to $\mathcal{D}_t^*(0) = 3$. More generally, the feed-forward depth d_f is commonly the same as $\mathcal{D}_t^*(0)$ which in turn corresponds to the maximum length from an input to its current output.

- **Recurrent skip coefficient**. Although both the concepts of recurrent and feed-forward depth provide theoretical measures of the complexity of an RNN, we also need another measure expressing the ability of our model to potentially handle long-term dependencies. Indeed, as better described in Section 3.1.2, since models with large recurrent depths have more nonlinearities through time, gradients can explode or vanish more easily.

  As we have also seen that one method to mitigate these problems consists in adding skip connections through multiple time steps, the last measure specialises on such additional connections. To that extent, the focus is now on the length of the shortest path from time i to time $i+n$, as the gradients flowing back through that paths would propagate more easily. From the unrolled RNN graph $\mathcal{G}_{un}$, $\forall i,n \in \mathbb{Z}$, let $\mathfrak{d}_i(n)$ be the length of the shortest path. As similarly done for the recurrent depth, the growth rate of $\mathfrak{d}_i(n)$ is used.

  Theorem 3.4.3. *Given an RNN and its two graph representations* $\mathcal{G}_{un}$*,* $\mathcal{G}_c$*, the growth rate of* $\mathfrak{d}_i(n)$ *can be obtained as follows under mild assumptions:*

$$j = \lim_{n\to\infty} \frac{\mathfrak{d}_i(n)}{n} = \min_{\vartheta \in C(\mathcal{G}_c)} \frac{l(\vartheta)}{\sigma_s(\vartheta)} \tag{3.69}$$

For convenience, considering that it is often the case that j is less than or equal to 1, the recurrent skip coefficient is defined as the reciprocal of j (i.e. $s = j^{-1}$). In sum, a large coefficient indicates that the number of transformations per time steps is smaller. In this case, the skipped recurrent connection would make it easier to let information flow across multiple time steps, which in turn can facilitate learning long-term dependencies. Nevertheless, not all types of skip connections can increase the recurrent skip coefficient.

The ability to define a model and compute the architectural complexity measures for an RNN is certainly beneficial. Besides the advantage of presenting a more theoretical picture, understanding how these complexity measures can affect the performance of a recurrent network is also an essential asset for explaining experimental results. Indeed, S. Zhang et al. (2016) demonstrated that these measures are related to the performance of an RNN; the main findings of their experiments can be summarised by the following considerations.

- Introducing additional recurrent connections, or changing the direction of the existent ones, can improve the performance of an RNN as long as we increase the recurrent depth.

- Increasing both the feed-forward and the recurrent depths d_f, d_r can aid in better capturing the over-time non-linearity of sequences. However, for too large d_r (or d_f), increasing d_f (or d_r) hinders the performance of the resulting models. This may be attributed to the optimisation issues typically arising when modelling large input-to-output dependencies.

- From a closer investigation on the impact of the feed-forward depths, the authors claim that d_f can only capture the *local nonlinearity* and has less effect on the long term prediction. This consideration contradicts previous claims (Hermans & Schrauwen, 2013) that stacked RNNs ($d_f > 1, d_r = 1$) could capture information in different time scales and would thus be more capable of dealing with learning long-term dependencies.

- A large recurrent depth d_r indicates multiple non-linear transformations per time step, which in turn results in potentially greater gradient vanishing/exploding issues. This suggests that d_r should be neither too small nor too large.

- The ways in which skip connections are designed makes significant differences on the performance of an RNN. More precisely, when comparing different models with skip connections – all crossing the same time length, only the model with a larger recurrent

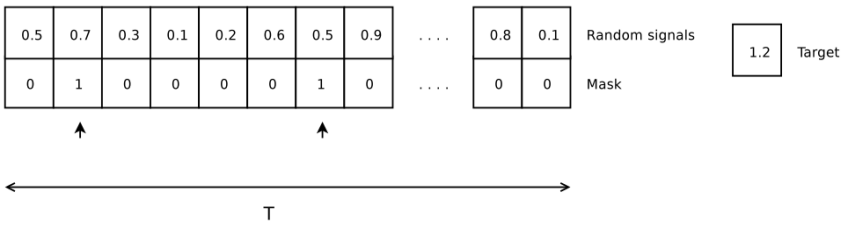


Figure 3.20: Illustration of the adding problem. Source (Le et al., 2015).

skip coefficient is expected to perform better. Therefore, when addressing long-term dependencies using skip connections, instead of only focusing on the time intervals crossed by our skip connection, we should also consider the model's recurrent skip coefficient, which can serve as a guide for introducing more powerful skip connections.

3.4.2 Evaluating the memory capabilities of RNNs

Whenever a novel architecture for sequential processing is introduced, it is good practise to test the model on a suite of commonly accepted sequence learning problems, and compare the evaluations with state of the art methods and baselines. Altogether, these tasks implicitly define a framework for measuring the performance of RNNs on a variety of tasks – both synthetic and real-world problem in contextually relevant domains. Among these tasks, some are specifically considered or designed to stress the ability of models to capture long-term dependencies.

This commonly accepted testing framework for RNNs typically comprehends a collection of sequence learning tasks which can be organised in four main categories, depending on the nature of these problems/tasks: *synthetic benchmarks*, *image classification*, *polyphonic music modelling* and *language modelling*. Each of these task is outlined in the following subsections, with the exception of music modelling, given that Chapter 4 is fully dedicated to the topic.

As the name suggests, *synthetic tasks* are not necessarily real-world problems. Typically, they can be easily solved with simple algorithmic procedures, although they are not as trivial for neural networks. The most popular examples are the *adding* and the copy memory problems.

- **The adding problem**. The adding problem has been extensively used as a stress test for RNNs. Formulated as a sequence regression problem, the goal is to predict the sum of two numbers in a sequence of random signals, which are sampled from a uniform distribution in $[0, 1]$. As in Figure 3.20, at every time step, the input consists of a random signal and a mask signal. The mask signal has a value of zero at all time steps except for two steps when it has values of 1 to indicate which numbers should be added (Le et al., 2015).

  Considering the source of random numbers, a simple baseline would always return 1 regardless of the inputs. This will give a Mean Squared Error (MSE) of around 0.1767.

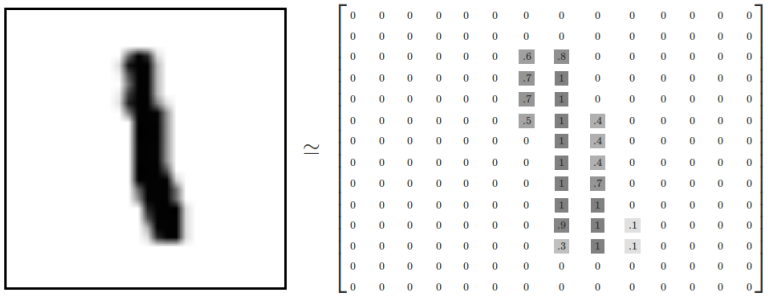


Figure 3.21: Example of MNIST input from (Abadi et al., 2015).

The problem gets harder as the length of the sequence T increases, as the dependency between the output and the relevant inputs becomes more remote. To solve this problem, the network must remember the first number or the sum of the two numbers accurately whilst ignoring all of the irrelevant numbers. This is the reason why researchers measure the performance of their models on increasing values of T for this synthetic problem.

- **The copy memory problem**. This problem measures the ability of a network to perform a simple operation that explicitly requires retaining large temporal dependencies. The problem is described as follows: each input sequence has length $T + 20$, where the first 10 values are chosen uniformly from the interval $[1, 8]$; the rest of the sequence contains all zeros, except for the last 11 entries that are filled with the digit 9 (used as a delimiter). The model is expected to produce zero outputs at every time step except for the last 10 entries, where it should generate (copy) the first 10 values in the same order as it has seen at the beginning of the sequence. As the goal is to minimise the average cross entropy of category predictions at each time step, we can observe how this problem gets more and more difficult as we increase the value of T, in a similar manner to the adding problem.

The second category of sequence learning problems is that of *image classification*. The most prominent dataset for this task is the MNIST dataset (LeCun, 1998), which contains 28×28 pixel images of handwritten digits (from 0 to 9) with the goal being the prediction of the digit represented in the image. As we can observe, considering the nature of image data, we are not dealing with pure sequential data. Nonetheless, the two following learning tasks are based on the intuition of treating images sequentially.

- **Sequential MNIST**. To turn the digit classification problem of MNIST into a sequence classification task with long-term dependencies, we can reshape image data into a 784×1 sequence. The resulting sequence learning problem is known as Sequential MNIST (Arjovsky, Shah, & Bengio, 2016; Le et al., 2015). In other words, an RNN model reads the sequence *one pixel at a time* and in turn predicts the digit represented in the image.

If we think of the way a single character is represented (Figure 3.21), the pixels at the top and bottom of each image are empty. As a result, when the images are arranged into a sequence of pixels, all the important pixels will be in the middle of the sequence. To exploit the informative value of that pixels, a recurrent network will need to form long-range dependencies that reach the middle of each sequence. We can thus agree that the model will have formed the necessary long-term dependencies when it outperforms a naive strategy of randomly guessing each digit (Le et al., 2015). As such, a naive strategy would achieve an expected accuracy of 10%, which is the baseline to beat.

- **Sequential Permuted MNIST** (pMNIST). This is a separate and more challenging task resulting from a slight modification of the Sequential MNIST dataset. More precisely, we permute the image sequences by a fixed random order beforehand, and the same permutation mapping must be used on each image to keep the data consistent between images. As pointed by Le et al. (2015), empirical results indicate that both *tanh* RNNs and LSTMs did not achieve satisfying performances, which also highlights the difficulty of this task.

Another common category of sequence learning tasks for testing RNNs is that of *polyphonic music generation*. As introduced in Section 2.1 and extensively discussed in Chapter 4, the task consists in training a model to predict the (group of) notes to be played together in the next time interval, given the previous ones. In the polyphonic settings, two elements characterise the performance of a model: temporal dependencies and chord conditional distributions. As argued in (Boulanger-Lewandowski et al., 2012), this task represents a well-defined framework to improve machine learning algorithms and is directly applicable to polyphonic transcription.

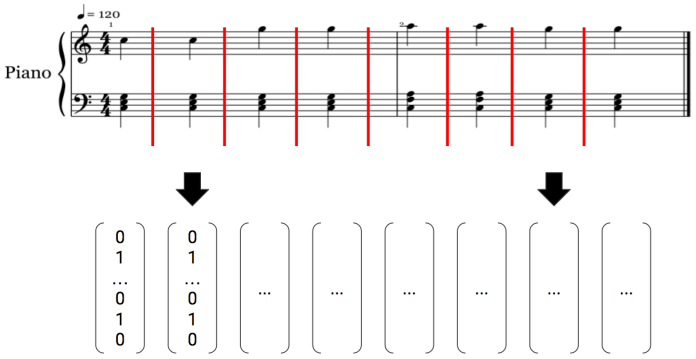


Figure 3.22: Example of piano-roll encoding, from `http://yoavz.com/music_rnn/`.

Rather than dealing with more complex, acoustically rich audio signals, music modelling in the context of RNN evaluation focuses on symbolic music – represented by the explicit timing, pitch, velocity and instrumental information. To give an idea of how music can be treated as a sequential process, Figure 3.22 shows an excerpt of a piano-roll representation – a binary matrix

encoding the active notes at each time step. The performance of a polyphonic music model is generally measured as a classification problem, hence in terms of negative log-likelihood.

The last category of tasks within our arsenal of benchmark tasks is that of *language modelling*, which probably represents the most common type of stress test for sequence models. The tasks per se is akin to music modelling, but with text instead of music.

As introduced in Section 3.2.2, an RNN can implement a character- or a word-level language model, depending on the desired granularity. Whereas an RNN trained as a character-level language model approximates the probability distribution of the next character, given the past characters, a word-level RNN achieves the same at the word level. In this context, quality of fit is often evaluated by perplexity for word-level language models and by the bits-per-character (BPC) metric (i.e. the binary logarithm of perplexity) for character-level models.

Some of the most common datasets or *corpora* used for language modelling are listed below.

- **PennTreebank** (PTB) (Marcus, Marcinkiewicz, & Santorini, 1993) is a corpus that has been extensively used in computational linguistics. If used as a character-level language corpus, it contains more than 5M characters for training, 396K for validation, and 446K for testing, with an alphabet size of 50 (Bai, Kolter, & Koltun, 2018). When used as a word-level language corpus, PTB contains 888K words for training, 70K for validation, and 79K for testing, with a vocabulary size of 10K. Despite its popularity, PTB is a relatively small language modelling dataset.

- **Wikitext-103** (Merity, Xiong, Bradbury, & Socher, 2016) contains 28K Wikipedia articles (about 103M words) for training, 60 articles (about 218K words) for validation, and 60 articles (246K words) for testing. It is almost 110 times as large as PTB, featuring a vocabulary size of about 268K. Hence, Wikitext-103 is more representative than PTB, as it also features a much larger vocabulary that includes many rare words.

- **LAMBADA** is a corpus containing 10K passages extracted from various novels, and with vocabulary size of about 93K. Introduced in (Paperno et al., 2016), LAMBADA was designed for the task of guessing the missing word in a sentence, given a number of the context sentences. Each sequence has an average of 4.6 sentences as context, and 1 target sentence, the last word of which is to be predicted. Due to the complexity of this word-level language task, performance also expresses the ability of a recurrent network to capture information from longer and broader contexts.

- **text8** contains 100M characters from Wikipedia (90M for training, 5M for validation, and 5M for testing), with an alphabet size of 27 (number of characters). The dataset is about 20 times larger than PTB, and is generally used for character-level language modelling.

3.5 A taxonomy of Recurrent Neural Networks

This section introduces a taxonomy of RNN architectures that is obtained from a 2-step categorisation process: first, recurrent models are grouped according to the specific problem they seek to address; the resulting groups are further categorised based on the technical nature of their approach. The first categorisation gives us the main directions of improvement of the recurrent architectures. The groups emerging from the second categorisation step are useful to understand the methodological similarity of these contributions. In other words, the taxonomy is based on "the why" (the motivations, the advantages) and "the what" (the actual nature of the contribution) of RNNs, and it fosters their composability by presenting the minimal set of them.

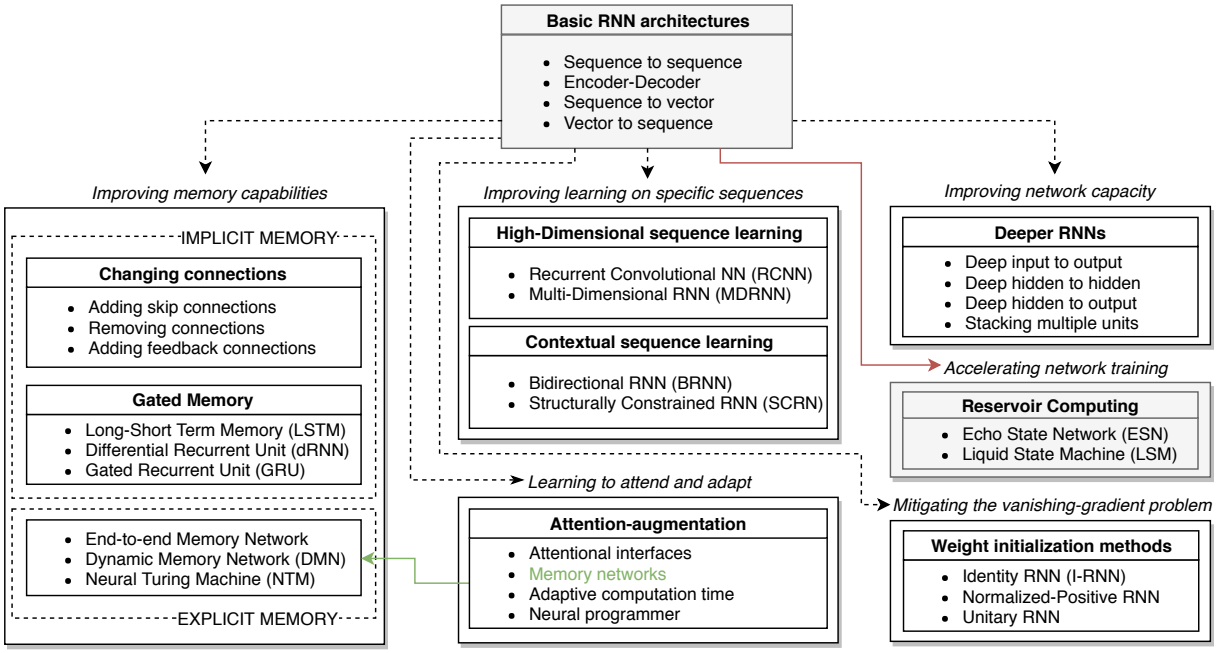


Figure 3.23: A taxonomy of RNNs, emphasising the composability of current approaches.

The proposed taxonomy is illustrated in Figure 3.23 and provide a useful resource emphasising the composability of RNNs. Machine learning practitioners can derive more complex recurrent models by combining, extending, or refining the basic architectures therein.

To explain the taxonomy, we assume to design a recurrent network incrementally, according to the **task** to perform, the main **objectives** we aim to achieve, and the **resources** available for the implementation of the model. The idea of this approach is to start with a basic RNN architecture, namely one of those presented in Section 3.1. This process is strictly dependent on task, i.e. *continuous sequence to sequence*, *delayed sequence to sequence*, *sequence to vector* and *vector to sequence learning*. This is the very first step to build our RNN. Next, to improve our baseline model, we can consider one or more steps from the following technical objectives.

- Improving the **memory capabilities** of our network, namely its ability to capture the context of long sequences, adapt dynamically in response to new contexts, as well as to accumulate knowledge and memorise facts explicitly. All these features can also help to capture long-term dependencies, thus mitigating the vanishing gradient problem. This can be achieved in either or both of these two ways.

  - **Implicit memory**, inherently embodied within the Recurrent Unit (RU) of the network and represented by the collection of all the hidden states. Therefore, to improve the memory capabilities of the RU we have again two ways to consider.

    * Changing the connections to and from the RU, such as: *adding skip connection through time* (adding direct connections from variables in the distant past to variables in the present); *removing length one recurrent connections* and replacing them with longer connections; *adding feedback connections* from the target units (if available) to the RU at training time.
    * Gating the information flowing in the RU to dynamically control the update of the hidden state. This approach introduces self-loops in the RU to produce paths where the gradient can flow for longer duration. The weights of such self-loops are gated (controlled by another hidden unit) in order to be conditioned on the context. Gating mechanism are implemented in *Long-Short Term Memory* (LSTM), *Gated Recurred Unit* (GRU) and *Differential Recurrent Unit* (dRU).

  - **Explicit memory**, akin to the concept of Random Access Memory (RAM) in the Von Neumann (computer) architecture, allowing processors to explicitly hold and manipulate pieces of information that are needed to perform a certain task. In the context of RNNs, explicit memory is achieved by *Memory Networks* (MemNN), which can use a set of memory cells by means of a supervised addressing mechanism. *End-to-end Memory Networks*, *Neural Turing Machines* (NTM) and *Dynamic Memory Networks* (DMN) overcame the need of supervision and can thus learn to *read* from and *write* arbitrary content to memory cells without explicit supervision – via the use of a content-based soft *attention mechanism* (Bahdanau et al., 2014).

- Improving the learning capabilities of the network on **complex sequences**, namely those sequences showing strong and non-trivial dependencies among observations on different dimensions or across multiple directions in the sequence. In this context, we can consider either or both of the following extensions.

  - Learning high-dimensional sequences, that is essential for all those problems requiring an understanding of contextual dependencies over multiple dimensions. For this

purpose, as a hybrid approach, the introduction of a Convolutional Neural Network (CNN) together with the incorporation of recurrent connections into each convolutional layer can shape a *Recurrent CNN* (RCNN). On the other hand, a *Multi-Dimensional RNN* (MDRNN) attempts to achieve the same purpose by adopting recurrent connections for each dimension at a higher computational cost.

  - Learning high-contextual sequences, that is a common requirement of recurrent networks dealing with sequences in which dependencies may not only be causal (from past to present). In this context, the output of the network at a given time step may depend on both the previous and next observations in the sequence. To achieve that aim, Bidirectional RNNs combine an RNN that moves forward through time beginning from the start of the sequence with another RNN that moves backward through time beginning from the end of the sequence (Goodfellow et al., 2016).

- Improving the **learning capacity**, that is generally defined as the ability of a network to model more complex functions. To that aim, Pascanu, Gulcehre, Cho, and Bengio (2013) argued that adding one or more nonlinear layers in the transition stages of an RNN can improve overall performance by better disentangling the underlying variations of the input. Deeper computation can be introduced in the *input to hidden*, *hidden to hidden*, and *hidden to output* parts. Another common approach to increase network capacity is simply to *stack multiple recurrent units* (layers) on each other.

- Mitigating the **vanishing gradient** problem by weight initialisation methods. The recurrent networks resulting from the application of this family of approaches seek to avoid the vanishing problem by design. In particular, the network is forced to stay in a region of the parameter space where the gradients do not vanish or explode.

- Accelerating the **training time**, by considering a different family of recurrent networks, implementing the paradigm of reservoir computing. This approach consists in creating a collection of recurrently connected units (called reservoir) and initialising the weights of their connections in a clever way. The high-level idea is to perform this initialisation process in such a way that the recurrent hidden units can capture the history of past inputs, so that we only need to learn the output weights during training. This idea was independently proposed for *Echo State Networks* (ESN) and *Liquid State Machines* (LSM).

  Nonetheless, it is important to remark that not all the architectural extensions discussed before would still be applicable in this case (note the red arrow in Figure 3.23).

Because of the grand challenges in modelling sequential data, the literature has been always evolving towards the search on novel architectures and methodologies to achieve (and even

outperform) human-level performance in both specific and general domains. Considering the scope of the thesis, this section focuses on the currently more general recurrent models and methods in the literature, i.e. gated recurrent memories, explicit memory networks, and how the use of attention mechanisms can be adapted to benefit different aspects of the RNNs in favour of their performance as well as the computational sustainability of such deep models.

Therefore, for the sake of our investigation, the rest of this section is dedicated to expanding the left-most branch of the taxonomy in Figure 3.23, which comprehends all those models aiming at improving the memory capabilities of RNNs, including attention.

3.5.1 Extending recurrent units for improving implicit memory

As already outlined in Section 3.1.2, the first attempt to improve the memory capabilities of a recurrent network – with the consequent benefit of mitigating the vanishing gradient problem, consists in extending the basic recurrent unit of vanilla RNN. The goal is to retain more information/context from the past elements of a sequence. This can already be done in several ways, among which: accumulating a running average of the previous hidden states (i.e. leaky units), adding skip connections through time, and replacing standard recurrent connections with longer connections to explicitly force units to operate on a long time scale.

Besides the former approaches, we can identify a family of more sophisticated recurrent units sharing the same motivation and the conceptual organisation of leaky units. These are commonly knows as *gated memory networks* and *gated RNNs* – which include the long short-term memory (LSTM) as well as the gated recurrent unit (GRU). The latter architectures are both state of the art methods in sequence modelling.

Gated Memory Networks

Similarly to leaky units, gated RNNs are based on the idea of creating paths through time that have derivatives that neither vanish nor explode (Goodfellow et al., 2016). Whereas leaky units address this with connection weights that were either manually chosen constants or treated as parameters, gated RNNs use connections that can contextually adapt at each time step.

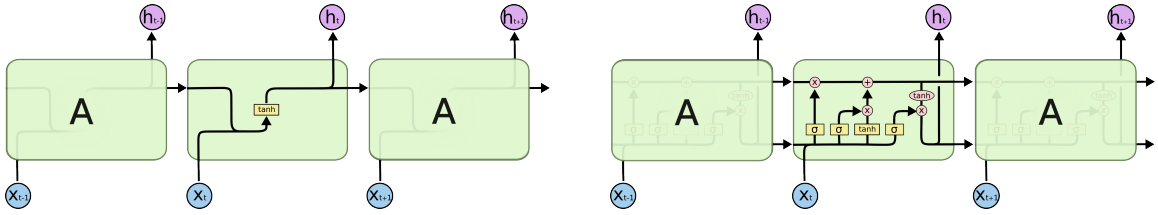


Figure 3.24: Simple unfolded view of an RNN layer vs an LSTM layer. Source: (Olah, n.d.).

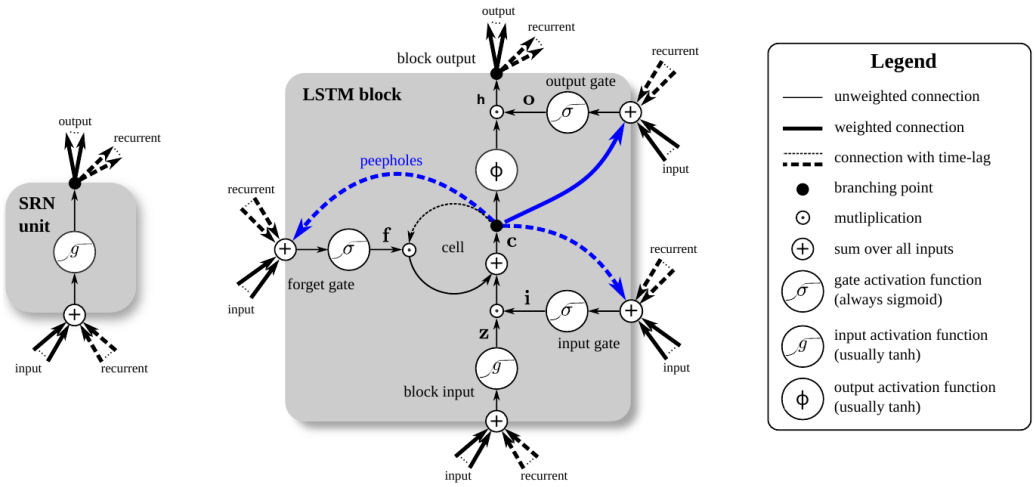


Figure 3.25: Comparison of the Simple RNN unit with the Vanilla LSTM block. Source: adapted from (Greff et al., 2017).

The most popular type of gated memory is the **long short-term memory** (LSTM) model (Hochreiter & Schmidhuber, 1997), which can also be viewed as a generalised framework for creating more specialised gated units. The core idea of the LSTM is to create self-loops in the recurrent unit so as to introduce paths where the gradient can effectively flow for long duration (without vanishing nor exploding). As reported in Figure 3.24, instead of a unit that simply applies an element-wise non-linearity to the affine transformation of inputs and recurrent units, an LSTM network contains a number of "LSTM cells" which in turn have an internal recurrence (i.e. a self-loop), in addition to the outer recurrence of the whole RNN.

More precisely, each LSTM cell has the same inputs and outputs as an ordinary recurrent network, but uses additional parameters and a clever system of gating units controlling the flow of information within the unit (Goodfellow et al., 2016). Conversely to traditional recurrent networks, each LSTM cell holds a personal state unit $c_i^{(t)}$ which update is regulated by two context-dependent gating mechanisms: one for the forget rate of the current value, and the other one controlling the extent of novel information to accumulate during the update. In other words, the extent to which the existing memory is forgotten is modulated by a *forget gate*, and the degree to which the new memory content is added to the memory cell is modulated by an *input gate* (Chung, Gulcehre, Cho, & Bengio, 2014). An additional *output gate* is used to regulate how much the state unit vector $c^{(t)}$ is exposed to the rest of the network.

Considering that several different versions/extensions have been proposed since the advent of the first LSTM by (Hochreiter & Schmidhuber, 1997), here we outline the most common LSTM setup used in literature; known under the name of *Vanilla LSTM* (Graves & Schmidhuber, 2005), it incorporates recent changes and uses full gradient training.

As can be seen in Figure 3.25, a detailed overview of the inner schematic of a vanilla LSTM

block reveals several differences with respect to a traditional RNN unit. Overall, a vanilla LSTM block features three gates (input, forget, output), block input, a state vector, an output activation function, and peephole connections. Moreover, to achieve a recurrent behaviour, the output of the block is connected back to the block input and all of the gates.

To formally describe how information flows within an LSTM block, we recall and introduce some useful notation, which is chosen to be consistent with that adopted in literature (and in particular with (Greff et al., 2017)). Let $x^{(t)}$ be the input vector at time t (or simply, the t-th element of the sequence under processing), N the number of LSTM cells in the block and M the number of inputs (i.e. the dimension of the input vector). It is also convenient to denote each element in the LSTM block with a letter; hence we denote the input block and the input, forget, output gates with letters z, i, f, o respectively. Then, considering the so-defined notation, a vanilla LSTM layer has the following parameters:

- Input weights: W_z, W_i, W_f, $W_o \in \mathbb{R}^{N \times M}$
- Recurrent weights: R_z, R_i, R_f, $R_o \in \mathbb{R}^{N \times M}$
- Peephole weights: p_z, p_i, p_f, $p_o \in \mathbb{R}^N$
- Bias weights: b_z, b_i, b_f, $b_o \in \mathbb{R}^N$

All these parameters need to be learned during training, and the role of each element in the LSTM block is suddenly clear if we describe the forward pass as follows.

1. As context is expected to change while processing an input sequence, the first step within the LSTM is to decide how much information we need to forget from the state vector $c^{(t-1)}$. This decision is made by a sigmoid layer called *forget gate*. More precisely, this gate looks at the current context given by: the previous LSTM state $c^{(t-1)}$, the previous LSTM output $h^{(t-1)}$, the current input vector $x^{(t)}$, according to the weights of its connections (i.e. W_z, R_f, p_f, b_z), and applies a sigmoid to the resulting affine transformation. The output is thus a vector of numbers in $[0,1]$, where each element is addressed to the cell state of a single LSTM unit $c_i^{(t-1)}$. In other words, a value close to 1 would not alter significantly a cell state, whereas a value of 0 would correspond to completely erasing/-forgetting a cell state. More formally, the forget gate step can be defined by this equation:

$$f^{(t)} = \sigma(W_f x^{(t)} + R_f h^{(t-1)} + p_f \odot c^{(t-1)} + b_f) \tag{3.70}$$

2. The next step is to decide how much new information we are going to propagate and store in the state vector. In the vanilla LSTM this process is made of two parts. First,

another context-based sigmoid layer decides which values are updated. This layer is called *input gate*, and the way it works is completely identical to that of the forget gate introduced before. Next, an additional layer called *block input* applies a context-based transformation to the input vector (i.e. a vector of candidate values for the new state $c^{(t)}$) before passing it to the input gate. If we denote an input activation function with $g()$ (usually a hyperbolic tangent function), this step is formalised by the following equations:

$$i^{(t)} = \sigma(W_i x^{(t)} + R_i h^{(t-1)} + p_i \odot c^{(t-1)} + b_i) \tag{3.71}$$
$$z^{(t)} = g(W_z x^{(t)} + R_z h^{(t-1)} + b_z) \tag{3.72}$$

3. Now that we know how much we need to forget from the previous state as well as how much and which new content we have to retain, we can proceed to update the state vector. Therefore, the LSTM block applies the forget gate to the old state, by means of the Hadamard product, and the same procedure is considered for the input gate and the transformed (or contextualised) input to obtain the new state vector as a sum of the resulting values. The formal description of this procedure is given by:

$$c^{(t)} = z^{(t)} \odot i^{(t)} + c^{(t-1)} \odot f^{(t)} \tag{3.73}$$

4. The very last step of the LSTM workflow consists in regulating the amount of state to expose to the rest of the neural network as output of the recurrent layer. The output of the LSTM can thus be defined as a filtered version of the state vector. Again, to achieve this design objective, a sigmoid layer called *output gate* modulates the amount of memory content exposure (Chung et al., 2014); in other words, it decides the strength of the filter w.r.t to every single element in the state vector. Finally, we put the state vector through an output activation function $\phi()$ (which is usually chosen to be a tanh) so that we can multiply it by the output of the output gate. More formally we compute:

$$o^{(t)} = \sigma(W_o x^{(t)} + R_o h^{(t-1)} + p_o \odot c^{(t-1)} + b_o) \tag{3.74}$$
$$h^{(t)} = \phi(c^{(t)}) \odot o^{(t)} \tag{3.75}$$

In accordance with the current conventions in the literature, the architecture of the vanilla LSTM described above also includes peephole connections (connections from the state vector to the gates, blue in Figure 3.25). As previously explained for the forget gate (although the same

consideration applies for all the other gates in the block) peephole connections are also part of the context under which the network has to respond and adapt by means of these powerful gating mechanisms. In other words, as argued by Gers, Schmidhuber, and Cummins (1999), peephole connections let the state vector control the gates thereby making precise timings easier to learn.

In sum, unlike the traditional recurrent unit which overwrites its content at each time-step, an LSTM block is able to decide and regulate whether (and by how much) to keep the existing memory via the introduced gates (Chung et al., 2014). For instance, if an LSTM cell detects an important feature from an input sequence at early stage, it easily carries that information over a long distance, hence, capturing potential long-distance dependencies.

From a number of experiments including real and artificial tasks (c.f. Section 3.4.2), LSTM recurrent networks have been demonstrated to learn long-term dependencies more easily than the simple recurrent architectures. Nevertheless, an LSTM block can be rather expensive to train because of the many parameters introduced. In an attempt to reduce the number of weighted connections in the memory block, Cho, Van Merriënboer, Bahdanau, and Bengio (2014) introduced the **gated recurrent unit** (GRU) – a simplified variant of the LSTM.

The main difference from the LSTM is that a GRU does not hold a separate state vector, and both the forget and input gates are replaced by a single gating unit which simultaneously controls the forgetting factor and the decision to update the state unit. Therefore, similarly to the simple recurrent unit, the whole state of the memory block is exposed to the rest of the network and peephole connections are not useful in any way. More formally, the update rules of a gated recurrent unit are described by the following equations:

$$h^{(t)} = u^{(t)} \odot h^{(t-1)} + (1 - u^{(t)}) \odot \tilde{h}^{(t)} \tag{3.76}$$

$$\tilde{h}^{(t)} = g(W_h x^{(t)} + R_h(r^{(t)} \odot h^{(t-1)}) + b_h) \tag{3.77}$$

$$u^{(t)} = \sigma(W_u x^{(t)} + R_u h^{(t-1)} + b_u) \tag{3.78}$$

$$r^{(t)} = \sigma(W_r x^{(t)} + R_r h^{(t-1)} + b_r) \tag{3.79}$$

where we used the same naming conventions presented for the LSTM, and vectors u, r respectively denote the update and reset gates of the GRU. From these equations, it is interesting to note how these new gates can individually ignore parts of the state vector (which is now fully exposed). In particular, the update gate acts like a contextualised version of the leaky integrator which can linearly gate any dimension to different extents, i.e. at one extreme, it can choose to completely ignore the present information and copy (or retain) the old value; at the other extreme, it can ignore the past by replacing the state vector with the new target state, so as to respond to an immediate context change. Additionally, the reset gates control which parts of

the state get used to compute the next target state, introducing an additional nonlinear effect in the relationship between past state and future state (Goodfellow et al., 2016).

Conceptual comparison of gated memories

To compare LSTM with GRU, we first observe that the most prominent feature shared between these units is the *additive component* of their update, which is lacking in the traditional recurrent unit. In fact, a simple recurrent unit always replaces its previous activation with a new value computed from the current input and the previous hidden state; this is somehow equal to fixing the forget gate of the LSTM to 1 (always forget implies always replace) or the update gate to 0. Conversely, both LSTM and GRU units keep updating the existing content in a flexible and adaptive manner; the additive nature of the state update has the at least two advantages.

- It is possible for each unit to remember the existence of a specific feature in the input stream even after several time steps: any important feature, decided by either the forget gate of the LSTM unit or the update gate of the GRU, will not be overwritten but be maintained as it is, if context has to preserved during sequence processing.

- The additive update creates shortcut paths that bypass multiple time steps and this is beneficial for natural gradient flow. More precisely, these connections allows the error to be back-propagated easily without too quickly vanishing (if the gating unit is nearly saturated at 1). Therefore, passing through multiple, bounded nonlinearities, reduces the difficulty of training the recurrent network due to possible vanishing gradients.

On the other hand, though we can consider a GRU as a simplified and lightweight version of the LSTM unit, it is also worth remarking their differences together with the implications arising from them. According to Chung et al. (2014), two distinctive elements can be identified:

- As can be noticed from their formalisation, a feature that the GRU misses from the LSTM unit is the contextualised gate-based exposure of the memory content. Indeed, the LSTM's hidden state at time t can be considered as the concatenation $(h^{(t)}, c^{(t)})$. Therefore, as pointed in (Jozefowicz, Zaremba, & Sutskever, 2015), an LSTM has two kinds of hidden states: a "slow" state $c^{(t)}$ that tackles the vanishing gradient problem, and a "fast" state $h^{(t)}$ that allows the unit to make complex decisions over short periods of time.

  Conversely, whereas the LSTM controls the amount of the memory content that is seen, or used by other units in the network via the output gate, a GRU exposes its full content without any control. It is thus reasonable to ask ourselves whether keeping a separate hidden state and a mechanism for controlling its exposure is truly beneficial or not. From

the extensive empirical study by Jozefowicz et al. (2015), it turns out that the output gate is often the least important for the performance of an LSTM: when removed, $h^{(t)}$ simply becomes $\tanh(c^{(t)})$ (for a tanh output activation function) which is generally sufficient for retaining most of the performance of an LSTM. This result can thus explain the comparable performance of GRU when compared to the LSTM.

- From a comparison of Equations 3.72 and 3.77, another difference is the location of the input gate, or the corresponding reset gate. More precisely, an LSTM unit computes the new memory content via the block input mechanism, which in turn lacks of a reset gate. Hence, there is no separate control of the amount of information flowing from the previous time step. However, the LSTM controls the amount of the new memory content being added to the memory cell independently from the forget gate (Chung et al., 2014).

  On the other hand, a GRU controls the information flow from the previous activation when computing the new, candidate activation, via the reset gate. In addition, considering the role of the update gate in regulating the tradeoff of novel vs old information, there is no independent control of the amount of the candidate activation being added. We can interpret this behaviour as a memory unit with limited retaining capabilities: "*if you need to store five bits of information, then I need to erase five bits of old data*".

From this brief comparison of LSTM and GRU, it is not possible to conclude which type of gating units would perform better in a general context. Overall, these units are both superior to traditional tanh unit and, in general, they perform comparably to each other.

On the optimality of the LSTM unit

The LSTM architecture is often criticised for being ad-hoc and having a substantial number of components/parameters whose purpose might not be fully clear (Jozefowicz et al., 2015). This observation also leaves us with the question of whether the LSTM is an optimal architecture per se, or if more efficient gated units exist. In this regard, Jozefowicz et al. (2015) investigated thousands of novel gated architectures by means of an extensive evolutionary architecture search. From their study it emerged that, although it is possible to find gated architectures that outperform both the vanilla LSTM and GRU on most tasks, LSTM units can effectively close this gap by using recurrent dropout and by adding a bias of 1 to the LSTM's forget gate.

The initialisation of the bias of the forget gate b_f is indeed an important technical detail to consider; if neglected, we may erroneously conclude that the LSTM is incapable of learning to solve problems with long-range dependencies. As observed in (Jozefowicz et al., 2015), initialising the LSTMs with small random weights would set the bias of the forget gate to 0.5, which in turn introduces a vanishing gradient with a factor of 0.5 per time step. Moreover,

as advocated by Gers et al. (1999) but rarely applied in several other studies, initialising this parameter to a value that is close to 1 would enable gradient flow.

Therefore, before attempting to use more sophisticated architectures, it is highly recommended to perform a careful initialisation of the LSTM's parameters. In addition, using specialised tools for black-box optimisation (Golovin et al., 2017) was demonstrated to boost the performance of vanilla LSTMs in such a ways as to outperform even more specialised architectures for language modelling (Melis, Dyer, & Blunsom, 2017).

3.5.2 Attention-powered recurrent neural networks

The concept of attention in humans and animals has been extensively studied in the fields of Cognitive Psychology (Oberauer, 2019) and Computational Neuroscience (Amso & Scerif, 2015). One of the most intuitive forms of attention is that of visual attention: if we think of the way we process visual information, we notice that we generally tend to focus on specific parts of the visual inputs we receive at a certain moment in order to "compute" the adequate responses. More precisely, according to the current context (e.g. what is one's state in relation to the current state of affairs, and which kind of actions/tasks one is expected to perform), there is a selection and filtration of the most pertinent pieces of information, as a large part of all the available information would potentially be irrelevant to compute the neural response.

In the field of machine learning, the concept of attention has also been modelled in artificial neural networks. As demonstrated by several studies, including attention models in ANNs has a large number of benefits: not only they can augment their effective learning capacity (i.e. the space of functions that can be approximated), but also the way they process information from one layer to another is more evident and clearer to visualise after training. In other words, attention mechanisms make it possible to understand which elements of the information available to a neural network, at a certain level of the computational graph and in different forms, are expected to be more relevant (or informative) in a specific context.

The use of attention in deep networks has brought huge benefits in many application domains, such as speech recognition, neural language translation, image captioning, and visual and non-visual question answering. Indeed, it is worth remarking that the current generation of deep networks – achieving state of the art results in several domains, make use of different variations of attention mechanisms in their architecture.

Despite the criticism on RNNs (Culurciello, n.d.), we can identify at least four different ways to include attention in the architecture of a recurrent networks: *layer-to-layer attentional interfaces*, *memory networks*, *adaptive computation time* and *neural programmers*. Before delving into each of these attention-augmented RNNs, we first need to provide the basic definition of an attention model for ANNs in general.

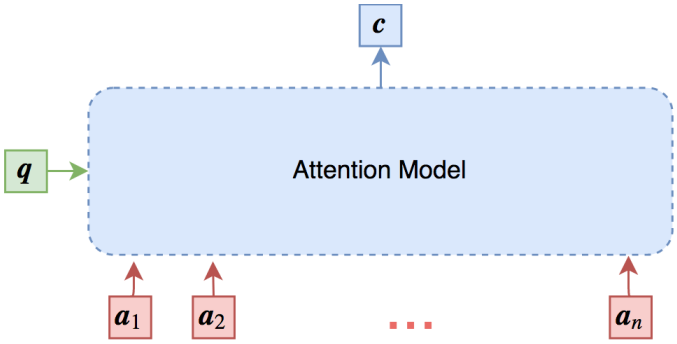


Figure 3.26: Representation of a general attention model.

As can be seen in Figure 3.26, an attention model is a general method that takes n arguments $a_1, \ldots, a_n$ called *annotations*, i.e. the object of attention, together with a *query* vector q providing the content w.r.t. which the annotation has to be attended. The model returns a vector c which is supposed to be the summary of the annotations focusing on those linked to the provided query vector q. In its basic form, it returns a weighted arithmetic mean of the annotations $a_1, \ldots, a_n$, where the weights (i.e. the attention vector) are computed in such a way as to reflect the importance/relevance of each annotation with respect to the query vector. The resulting summary vector c is also known as *glimpse* or *context*.

Therefore, an attention model in a recurrent architecture lets the network sequentially focus on a subset of some annotations, process it according to the recurrent units, and then change its focus to some other parts of that information. This procedure goes on as long as the network needs to be unfolded, which strictly depends on both the sequences to process and the task to perform. The attention model thus makes it easier to reason sequentially about the information we need to attend, even if that information is not sequential in nature.

The simplicity of the attention model makes it flexible enough to be applied in different stages and manners within the computational graph of a recurrent network. For instance, we can think of the annotations as continuous representations of our sequential input (e.g. word embeddings in case of a language model) and the context vector as the hidden state of a recurrent network processing that sequence at a higher level. Alternatively, the same approach can be applied among hidden layers, or even with respect to the cells of a memory bank which accumulates factual knowledge from the sequences under processing.

Towards a more formal definition of attention in neural networks, we recall that our objective is to obtain a context vector c that is a compressed representation of those annotations which are more relevant w.r.t. the query vector q. To that aim, we need to define a mechanism ϕ that computes c from the annotation vectors $a_1, \ldots, a_n$ according to their relevance to the query vector. More precisely, for each annotation a_i the mechanism should generate a positive weight

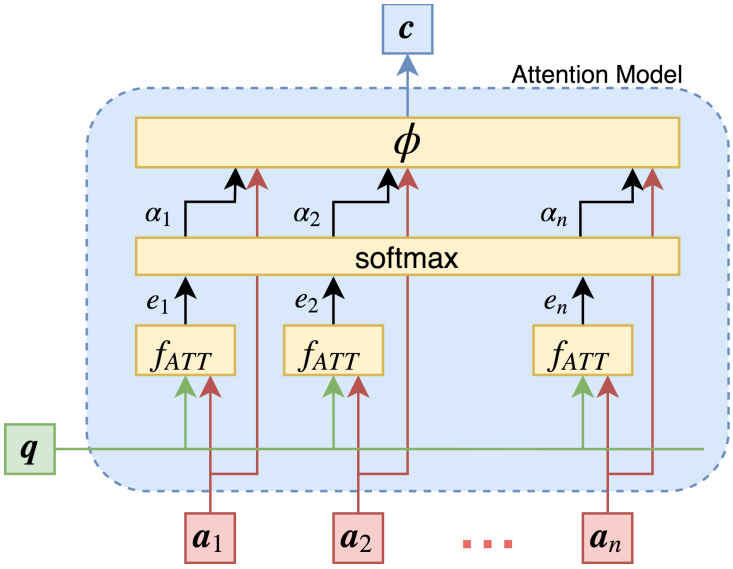


Figure 3.27: Internal view of a generalised attention model.

α_i which can be interpreted as either the probability that annotation a_i is the right content to focus on, or as the relative importance to give to annotation a_i in "blending" (or averaging) all the a's together. As better explained later on, the first interpretation gives the concept of *hard attention*, which is stochastic and non-differentiable, whereas the seconds one corresponds to *soft attention*, which is the preferred choice due to its deterministic and differentiable nature.

The weight α_i of each annotation vector a_i is computed by an attention model f_{ATT} which is often chosen to be a multilayer perceptron conditioned on the query vector q; the resulting vector of attention weights $\alpha \in [0,1]^n$ is called *attention vector*. In line with the generalised attention model illustrated in Figure 3.27, the attention vector as follows.

1. A score e_i is first computed for each annotation a_i according to a specific function f_{ATT} which is the core of the attention model. As mentioned before, the main purpose of this function is to capture the relevance of each a_i given q, according to the task-based design criteria of the modeller. Hence, higher scores translate to greater relevance, but only an investigation of the whole score vector can reveal whether an annotation would get high attention or not; this last consideration will be clearer after the second step discussed below. In the generalised model, a single score is computed as:

$$e_i = f_{ATT}(a_i, q) \tag{3.80}$$

The simplest version of that function consists in computing the dot product between each annotation vector a_i and the given query vector q, and eventually applying the hyperbolic tangent function (i.e. $f_{ATT} = \tanh(a_i q)$ or just $f_{ATT} = a_i q$). Such an independent vector

aggregation is also known as *content-based* attention. More sophisticated and combined forms of attention can thus be obtained by providing a specialised definition of f_{ATT} which both meet the requirements of the attention model and reflect overall design goals.

2. To finally compute attention, all the scores need to be squashed into the interval $[0,1]$ so that all the elements in the attention vector sum to 1. This can be easily achieved by means of the softmax function, which takes the score vector e and compute the attention vector α. The attention for the i-th annotation is thus given by:

$$\alpha_i = \text{softmax}(e)_i = \frac{\exp(e_i)}{\sum_{k=1}^{N} \exp(e_k)} \tag{3.81}$$

Once the attention vector α (with elements summing to one) is computed, the next and final step consists in the computation of the output of the attention model, i.e. the context vector c. Again, considering that the attention model presented here represents a general framework for different possible implementations, we describe the computation of the context vector as a function ϕ of the annotation vectors and their corresponding weights:

$$c = \phi(\{a_i\}_{i=1}^{N}, \{\alpha_i\}_{i=1}^{N}) \tag{3.82}$$

which returns a single vector c summarising the context of the annotations w.r.t. the attention values computed in the previous steps. Although different implementations exist, the literature (Bahdanau et al., 2014; K. Xu et al., 2015) indicates two main functions for ϕ.

- **Hard attention**, forcing the attention model to make a hard decision on which annotation vector to consider by sampling one of the annotation vectors following a categorical distribution. More precisely, a location variable s is introduced to indicate where the model decides to exclusively focus attention when the query vector q is presented to the model; we can think of each s_i as a *binary location indicator* (or one-hot variable) which is set to 1 only if the i-th annotation (out of N) has to be attended.

  By treating the attention locations as intermediate latent variables, we can assign a Multinoulli distribution parametrised by $\{\alpha_i\}$, and view c as a random variable:

$$p(s_i = 1 \mid a) = \alpha_i \tag{3.83}$$

$$c = \sum_i s_i \odot a_i \tag{3.84}$$

Therefore, hard attention is a stochastic process; instead of using all the annotations for producing a context vector (the glimpse), we decide to attend on annotation a_i with probability α_i. Due to the non-deterministic nature of hard-attention, the learning process is thus more complex. A common approach that is preliminary to propagate a gradient, is to first estimate that gradient by Monte Carlo sampling.

- **Soft attention**, allowing the context vector to capture all the annotation vectors but with the extent given by their attention values. In other words, instead of sampling the attention location, we can take the expectation of the context vector c directly. This corresponds to formulating a deterministic attention model by computing a soft attention-weighted annotation vector as expressed by the following equation:

$$c = \phi(\{a_i\}_{i=1}^{N}, \{\alpha_i\}_{i=1}^{N}) = \mathbb{E}_{p(s|a)}[c] = \sum_i \alpha_i \odot a_i \tag{3.85}$$

This simple and effective attention model, introduced for neural machine translation (Bahdanau et al., 2014), corresponds to feeding in a soft weighted context into the system using attention. The whole model is thus smooth and differentiable under the deterministic attention, so learning end-to-end can be achieved by using standard back-propagation.

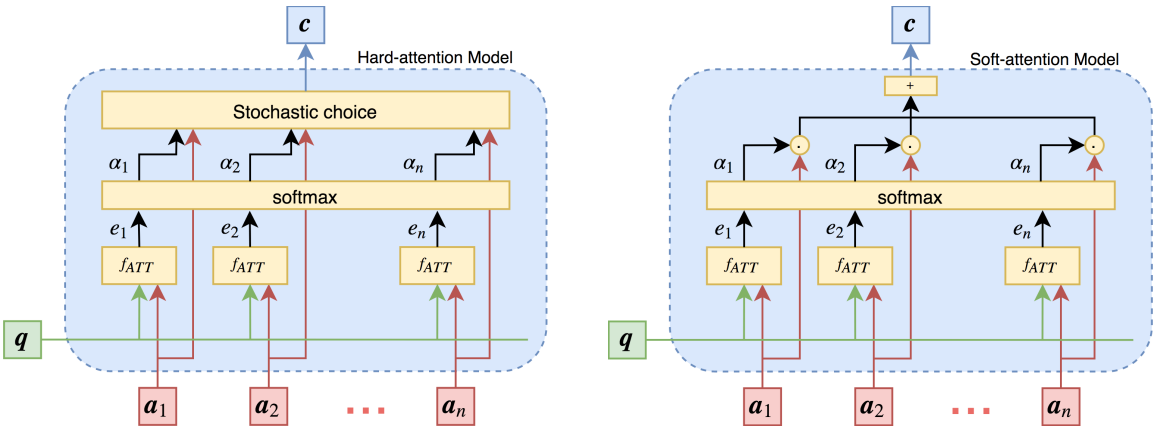


Figure 3.28: Hard attention (*Left*) vs soft attention (*Right*) general models

From this brief description of hard and soft attention mechanisms, we do not have enough arguments supporting the idea that one is better than the other. This kind of analysis is often performed once the learning task is clear and two neural network architectures are proposed for comparison: one with hard-attention and another one with soft-attention. Thereafter, a close investigation of the performance of the resulting models post-training, with special focus on how attention is distributed among the annotations from a human point of view, can reveal which of the two mechanisms is more natural and better suited for the task.

Nevertheless, as far as we can see from the literature in attention-augmented ANNs, the current trend is to use soft attention mechanisms due to their simple implementation, as the gradient is computed directly instead of being estimated through a stochastic process.

Now that the general definition of an attention model has been presented, a taxonomy of attention-augmented recurrent neural network in the context of sequence processing and generation can be presented. In particular, a critical analysis of how the general attention model presented above is adapted and used for different purposes in a recurrent architecture, groups the major contributions into four categories, i.e. *attentional interfaces*, *memory networks*, *adaptive computation time* and *neural programmers*. Individually, these techniques are all powerful extensions of RNNs, but they can be combined together (Olah & Carter, 2016).

Attentional interfaces

An attentional interface can be used to interface with a neural network that has a repeating structure (e.g. an image) or a sequential form (e.g. a sentence) in at least one of its layers. Indeed, if we visualise the architecture of a neural network with two recurrent layers, then we can identify at least three possible ways to implement attention.

1. Considering the sequential nature of input data, we can implement an attentional interface between the input and the first hidden layer. In this case, it is common practice to enrich input sequences in such a way as to present the network with continuous representations (e.g. word embeddings in case of textual data).

2. Introducing an attention model between two consecutive recurrent layers so that the top layer can focus on different positions in the other recurrent layer at every time step. In a more general context, this occurs when there is a layer acting as an encoder network, which can either process an input sequentially or not, and a recurrent layer sitting on the top as a decoder network. The latter, in turn, can produce alternative representations of the input focusing on the sequential or structured output of the encoder during this process.

3. Assuming that the network's output is sequential and the computation is recurrent, we can think of applying the same approach for attending to the output that is progressively generated from our network. Although the hidden states of a recurrent network are expected to retain past context, and considering that the last element of the so-far generated output sequence is fed back to the recurrent network (unless teacher forcing is used), an attention model could be used to provide more than a single past element to the recurrent layer expected to generate the output sequence.

The first type of attentional interface is probably the most intuitive, and can be directly implemented via the basic attention model presented before (Section 3.5.2). Instead, the *output-to-hidden* attentional interface can be considered as a special case of the second one; the only difference between these approaches lies in the increasing number of annotations to focus on at each time step. Hence, this part describes the second type of attentional interfaces, collecting all those architectures based on encoding and decoding linguistic and multimedia content with an attention model between encoder and decoder. To that aim, we outline the most popular architectures of attention-powered RNNs used for neural machine translation (Bahdanau et al., 2014), visual image captioning (K. Xu et al., 2015), as well as how the same methodology can be easily extended to deal with structured content in general (Cho, Courville, & Bengio, 2015).

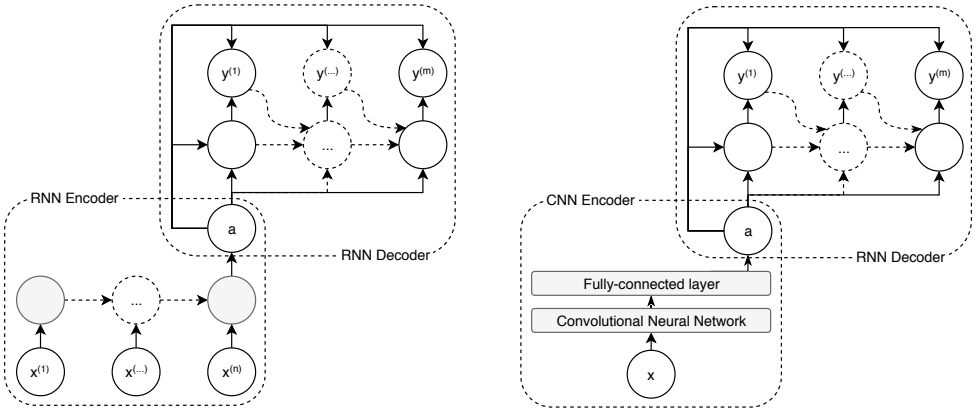


Figure 3.29: Illustration of the simplest and non-attentive forms of encoder-decoder architectures with recurrent (*Left*) and convolutional (*Right*) neural networks as encoders respectively.

As outlined in Section 3.1.1, the encoder-decoder baseline architecture is used to deal with structured output problems, where the task is to map the input to an output that possesses its own structure. Hence, the task is not only that of mapping the input to the correct output, but also to model the structure within the output sequence (Cho et al., 2015). For most of these problems, the structure of the output is intimately related to the structure of the input, and one of the main challenges is the problem of alignment, i.e. how to relate sub-elements of the input to sub-elements of the output. This is exactly what attention seeks to achieve.

Structured output problems represent a large and important category of problems, including both classic tasks such as machine translation, image captioning and speech recognition, as well as many natural language processing tasks (e.g. text summarisation and paraphrase generation) (Cho et al., 2015). To better understand how one can implement attentional interfaces in this context, a brief summary of the encoder-decoder framework introduced in Section 3.1.1 is given.

The encoder-decoder framework aims to provide an effective neural network architecture that possibly learns to capture the underlying, complex mapping between highly structured input

and output. As the name suggests, the overall architecture consists of an encoder and a decoder. The encoder implements a function f_{enc} which reads the input data x (e.g. a sequence of vector words encoded as embeddings or even an RGB image) into a continuous-space representation a called context or summary vector[4]. More formally, the output of the encoder is defined as:

$$a = f_{enc}(x) \tag{3.86}$$

As can be seen from Figure 3.29, the choice of f_{enc} largely depends on the type of input; when x is an image, a convolutional neural network (CNN) would be a trivial candidate. Conversely, a recurrent neural network is a natural choice when x is sequential.

The decoder then generates the output sequence y conditioned on the continuous-space representation, or summary a of the input. As outlined in (Cho et al., 2015), this is equivalent to computing the conditional probability distribution of the output sequence given x:

$$P(y|x) = f_{dec}(a) \tag{3.87}$$

In this setting, considering that our objective is to generate an output sequence providing an alternative representation of the input, while capturing the existent structural dependencies, a recurrent network is the standard choice to implement the decoder function.

Nevertheless, in the original implementation of the encoder-decoder framework, the continuous representation a of the input x returned by an encoder is a fixed-dimensional vector, regardless of the size of the input. In fact, when a recurrent network is used as an encoder, the last hidden state $h^{(m)}$ is typically considered as the summary vector. As a result, the fixed-size summary vector a is not structured by design, thus there is no guarantee that such a vector preserves the spatial, temporal or spatio-temporal structures of the input (Cho et al., 2015). For instance, in neural machine translation, the encoder needs to compress all the necessary information of a source sentence into the summary vector. Hence, the fixed-size nature of this compressed representation makes it difficult for the decoder to cope with long sentences, and this is especially true for those that are longer than the sentences in the training set (Bahdanau et al., 2014). As argued by (Cho, Van Merriënboer, Gulcehre, et al., 2014), the performance of a basic encoder-decoder deteriorates rapidly as the length of an input sentence increases.

Similarly, even when the size of the input is fixed, e.g. a fixed-resolution image, the amount of information contained in each image may vary significantly (Cho, Van Merriënboer, Gulcehre, et al., 2014). As an example, the task of image captioning is influenced by the number of objects in each image, which is not the same for every input. A fixed-size representation would

[4]However, from now on, we prefer to avoid the "context vector" denomination for the encoder's output because it is incompatible with the notion of the context vector (or glimpse) produced by an attention model.

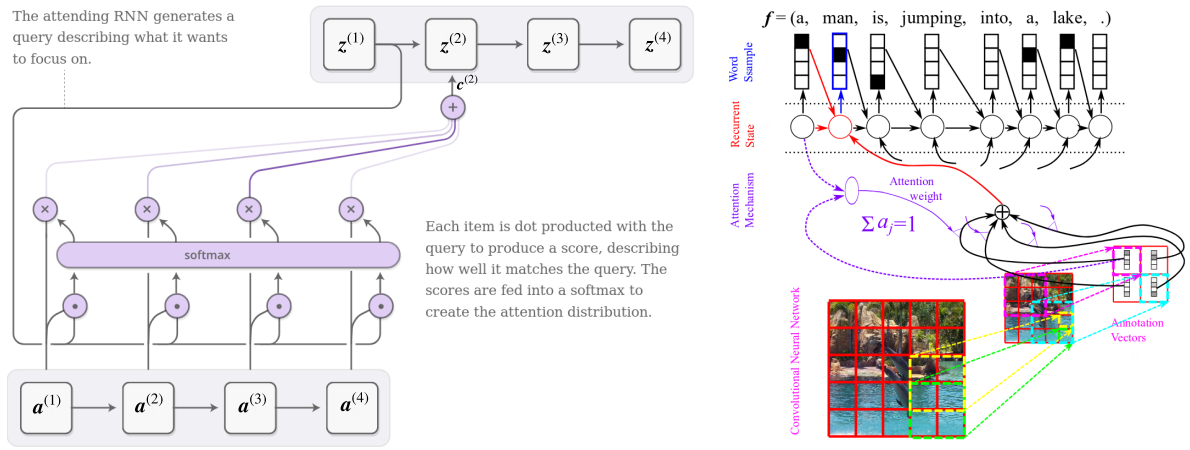


Figure 3.30: Attention-powered encoder-decoder models for machine translation (*Left*, adapted from (Olah & Carter, 2016)) and image captioning (*Right*, adapted from (K. Xu et al., 2015)).

instead compress every image in the same vector, and this may not be expressive enough to generate rich captions when the scene has a considerable number of pertinent objects/features.

These critical observations on the encoder-decoder framework allow us to understand that the use of an attentional interface among these layers is not only a powerful extension of this model (also improving the interpretability of the resulting trained network), but also a fundamental architectural component which makes it possible to overcome the need of a single fixed-size compressed representation of the input. More precisely, implementing an attentional interface for the encoder-decoder framework (and more generally, for a layer-to-layer architectural pattern, where the second layer is recurrent) requires us to consider the following steps.

1. **Free** the encoder from returning a fixed-length summary vector; instead, a structured representation of the input data is produced by allowing the continuous-space representation to be a *set* of fixed-size vectors, to which we refer as a summary set:

$$a = \{a^{(1)}, a^{(2)}, \ldots, a^{(m)}\} \tag{3.88}$$

where, according to the type of data we are dealing with, each vector in the summary set is localised to a certain spatial, temporal or spatio-temporal component of the input. To be consistent with Figure 3.29, we can distinguish the two following scenarios.

- In case of machine translation (Bahdanau et al., 2014), each context vector would summarise a phrase centred around a specific word in a source sentence. This is achieved through a bidirectional recurrent neural network (BiRNN) encoder.

A BiRNN consists of two recurrent layers: a forward recurrent that reads the input sequence as it is ordered (from $x^{(1)}$ to $x^{(m)}$) and calculates a sequence of forward hidden state vectors; a backward recurrent layer instead, reads the sequence in the reverse order (from $x^{(m)}$ to $x^{(1)}$), resulting in a sequence of backward hidden states. Finally, these vectors are concatenated per step to form a summary set. The following equations summarise the described flow.

$$\overrightarrow{h} = (\overrightarrow{h}^{(1)}, \ldots, \overrightarrow{h}^{(m)}) \tag{3.89}$$

$$\overleftarrow{h} = (\overleftarrow{h}^{(1)}, \ldots, \overleftarrow{h}^{(m)}) \tag{3.90}$$

$$a = [\overrightarrow{h}; \overleftarrow{h}] \quad \text{s.t.} \quad a^{(t)} = [\overrightarrow{h}^{(t)}; \overleftarrow{h}^{(t)}] \tag{3.91}$$

As advocated by Bahdanau et al. (2014), using BiRNN is crucial for capturing the contextual dependencies of each word w.r.t. the rest of the sentence, as each vector $a^{(t)}$ contains the summaries of both the preceding words and the following words.

- As for image captioning (K. Xu et al., 2015), instead of using the activation of the last fully-connected hidden layer as a single continuous-space representation of the input image, we use the activation from the last convolutional layer of a pre-trained convolutional network. Unlike the previous encoder model, the summary set consists of m feature vectors, each of which is a D-dimensional representation corresponding to a *specific spatial region* of the input image. This architectural trick thus allows to process an image sequentially and to use an attention model.

2. **Introduce** an attention layer between the revised encoder and the improved decoder we discuss later. In particular, we use the same basic form of attention model outlined in the preamble of the present section and illustrated in Figure 3.28. In this setting, the set of summary vectors $a = \{a^{(1)}, \ldots, a^{(m)}\}$ plays the role of annotations, whereas the hidden state of the decoder is presented as a query vector at each time step, i.e. $q^{(t)} = z^{(t-1)}$ (where $z^{(t-1)}$ denotes the hidden vector of the decoder at time $t-1$). As a result, the attention model can provide the decoder with a context vector $c^{(t)}$ (computed according to Equation 3.85) at each time step t, which in turn can be considered as a temporal glimpse of the annotations based on hidden state $z^{(t-1)}$.

   Moreover, it is worth observing that in most encoder-decoder models the hidden state of the decoder also corresponds to an element of the output sequence. Therefore, when generating a sentence, the previous outputted word is used as the query vector of the attention model for obtaining the context for the next word.

3. **Improve** the decoder in such a way as to make use of the attention model, and in particular, of the context vector we manage to obtain at each time step from the controlled attention model[5] in the previous layer. More precisely, we want the decoder to be a conditional recurrent layer that produces a sequence by generating one element at every time step t conditioned on a context vector $c^{(t)}$ (computed as before), the previous hidden state $z^{(t-1)}$ and the previously generated sequence element $y^{(t-1)}$. To implement this the context vector $c^{(t)}$ is used during the update of the hidden state of the decoder as follows:

$$z^{(t)} = f_{dec}(z^{(t-1)}, y^{(t-1)}, c^{(t)}) \tag{3.92}$$

and any form of recurrent neural network can be used to implement the decoder function f_{dec} according to the sequence to generate (an LSTM is often the standard choice). Similarly to the attentional interface explained before, the attention-powered version of our decoder can be reused for different structured output problems; therefore, the only strictly task-dependent aspect in this framework consists in the definition of an encoder which can provide a set of summary vectors from the input data.

In sum, the adoption of an attentional interface in the context of the encoder-decoder framework is beneficial for several aspects in the corresponding recurrent architecture. First, one can overcome the limitations arising from compressing any variable-length input into a single fixed-dimensional vector. This is done by spatially or temporally dividing the input so that the encoder can represent the input into a set of vectors, each of which encode a fixed amount of information focused around a particular region of the input (Cho et al., 2015).

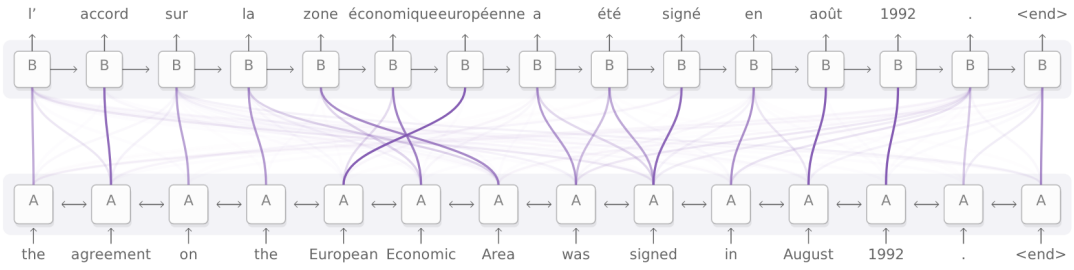


Figure 3.31: Example of attention alignments in machine translation. This allows us to see which positions in the source sentence were considered more important when generating the target word (Source (Olah & Carter, 2016)).

The second advantage of using attention is to allow the decoder to focus on distinct aspects, regions, or parts of the input while generating the output sequence. As argued by Cho et al.

[5] In the sense that the decoder hidden state is used to dynamically interact with the attention model by providing a query at each time step. We can thus consider the decoder as a controller of the interface.

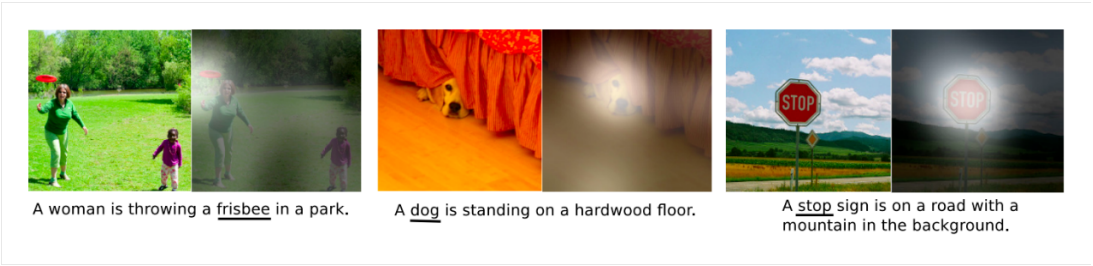


Figure 3.32: Example of spatial context used by the decoder while captioning: *white* indicates the attended regions corresponding to the *underlined* words. Source: (Olah & Carter, 2016).

(2015), this improves its ability to extract the most relevant information for each piece of the output, thus yielding improvements in the quality of the generated outputs.

As can be seen from Figures 3.31 and 3.32, the attention model can be easily inspected by visualising the annotation weights learned by the model. This provides an effective and intuitive way to *interpret the behaviour* of the model while outputting a sequence. Indeed, the magnitude of the attention weight α_j^t highly correlates with how predictive the spatial, temporal or spatio-temporal region of the input, to which the j-th summary vector corresponds, is for the prediction associated with the output element $y^{(t)}$ at time t. Not only can one inspect the alignment between the output sequence elements and the summary vectors by visualising the annotation weights, but can also exploit such visualisations to get an intuition as to why classification/generation mistakes were made (K. Xu et al., 2015).

Other than neural machine translation and image captioning, the attention-powered encoder-decoder framework is extensively adopted in structured output problems, and especially in the domain of multimedia description generation; the latter comprehends those problems aiming at generating a language description of a multimedia input, as well as text in another language. Examples of applications within this family of problems include neural speech recognition and video description generation (i.e. translating a video clip to its natural language description). A comprehensive literature overview of attention-powered encoder-decoder networks for describing multimedia content is provided in (Cho et al., 2015).

More generally, we can conclude that a layer-wise attentional interface like the one described so far, makes it possible to learn the underlying mapping between two entirely different representations without explicit supervision of that mapping. The learned alignments can also be visualised for interpretation, and they are usually found to agree with human intuition.

Memory networks

Most machine learning models lack a (potentially large) long-term memory component to *read from* and *write to* whenever they need to manipulate large amount of data and perform inference

from such information (Weston, Chopra, & Bordes, 2014). Nonetheless, one might also argue that a separate memory structure would not be necessary for certain tasks due to the different memory requirements associated with each learning problem. On the other hand, one cannot neglect all those tasks which explicitly require a factual memorisation of previous information; as an example, consider the problem of question answering (QA) in which a neural network is given a set of facts or a story, and it is expected to answer one or more questions on that subject. Therefore, how can it memorise and manipulate these facts?

In theory, this problem can already be addressed by RNNs, as they have been shown to be Turing-Complete (Siegelmann & Sontag, 1995), and therefore have the capacity to simulate arbitrary procedures. However, Weston et al. (2014) argued that what makes it difficult for RNNs to accurately remember facts from the past is due to the following considerations: (i) recurrent networks implement implicit memory, which is encoded in the form of their hidden states and weights; and (ii) memory is not compartmentalised enough as knowledge is compressed into dense vectors. As a result, even the more sophisticated models of RNNs, such as the gated memory networks described in Section 3.5.1, can struggle to fully memorise long sequences and learn simple tasks, like outputting the same input sequence they have just read. This is more evident for other tasks, e.g. in the vision and audio domains a more persistent and long memory is required to watch a movie and answer questions about it (Weston et al., 2014).

To address this problem, both the independent contributions of Weston et al. (2014) and Graves, Wayne, and Danihelka (2014) proposed a new powerful class of models called *memory networks*. Intuitively, memory networks extend the capabilities of neural networks by coupling them with an external memory bank and an attention model controlling the access to memory locations for read and write operations. The central idea is to combine the successful learning strategies developed in the ML literature for inference problems, with a separate memory component that can be dynamically accessed and manipulated. The resulting model can thus be trained to learn how to operate effectively with the new memory component.

To the best of our knowledge, since the introduction of this new class of attention-augmented neural network models, one can identify four major implementations of this idea. For the sake of clarity, each of these four differ in both the motivations and the design objectives of the memory component – in relation to the specific learning task to perform. The scope of this subsection is thus to present the conceptual idea of each contribution, so that we can critically analyse and compare these models according to their strengths and weakness.

- **Memory Neural Networks** (MemNN) (Weston et al., 2014) represent the first attempt proposing the integration of a memory component to a neural network for extending the

actual memory capacity. In this contribution, the authors provide both a *general framework* called IGOR for implementing memory networks, as well a specific model for question answering. The main limitations of this architecture are listed below. More precisely, considering that every sentence belonging to a text-question input pair is stored in the memory component, the proposed model:

- makes use of a hard-attention mechanism for retrieving the most relevant facts (i.e. sentences in the given text) needed for answering a certain question;
- requires strong supervision at each layer of the network, as we need to known a priori which are the actual relevant facts for every text-question pair.

As a consequence, a MemNN is not easy to train via back-propagation, because it is not differentiable end-to-end (due to hard-attention), and can only be applied if the supporting subset (i.e. the list of relevant sentences for each text-question input pair) is explicitly provided to the model during training. Nevertheless, it also interesting to observe that the neural network can access its memory component multiple times in order to retrieve several relevant facts before inferring an answer to a given question.

- **End-to-end Memory Networks** (MemN2N) (Sukhbaatar et al., 2015) are currently one of the most popular implementations of IGOR (Weston et al., 2014) for the specific task of question answering. The model can be seen as a continuous form of the MemNNs introduced before. The continuity of this model means that it can be trained end-to-end from input-output pairs, which is made possible by the use of a soft-attention mechanism for memory access. Moreover, the model must deduce by itself (both at training and test time) which sentences are relevant for answering a certain question. These two considerations are thus responsible for the extended applicability of the MemN2N model to other tasks, especially for those where supervision is not available.

  The main intuition of MemN2N is that of storing all the sentences of a text-question(s) input pair in a separate memory; we can think of this last persistent component as a matrix where each row is a dense representation (e.g. a bag or words) of the corresponding sentence. The process of generating an answer from a given question q is enabled by multiple memory accesses, each of which is based on a content-based attention mechanism controlled together by q and the output of the previous access. This allows the network to progressively extract the relevant facts from the memory to produce a final answer.

  However, the authors explicitly mention that such an iterative memory access is made possible by "*a novel recurrent neural network (RNN) architecture where the recurrence reads from a possibly large external memory multiple times before outputting a symbol*"

(Sukhbaatar et al., 2015). Instead, as one can see from the two architectures reported in Figure 3.33, a MemN2N is not a recurrent network per se, since the number of memory accesses to perform (which we can consider as the length of the recurrence) before generating an answer is fixed by design. This results in a feedforward neural network with attention-based memory access, where the number of memory hops is equal to its feedforward depth minus 1 (because of the fully connected softmax layer for answer inference). Alternatively, the MemN2N is in fact a recurrent network which is unfolded for a fixed number of time steps to enable the model to access its memory bank multiple times.

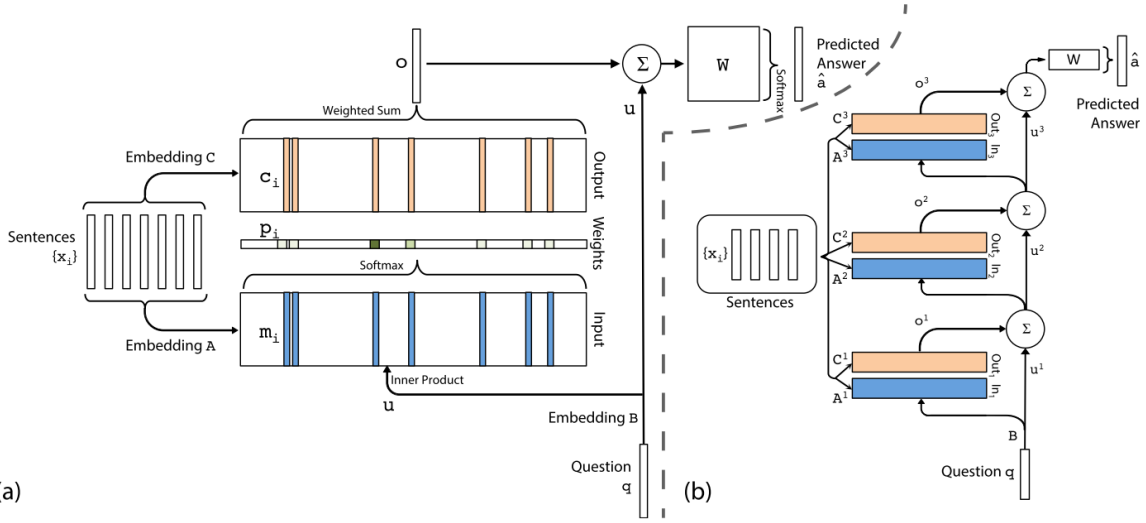


Figure 3.33: Architectures of a single-layer MemN2N (a) and a three-layer MemN2N (b) from (Sukhbaatar et al., 2015). The fixed structure imposes a single memory access for the first model and three accesses for the second one.

As we are not dealing with an RNN, a MemN2N does not process the input sentences and the corresponding questions in a sequential manner, nor it generates an answer word by word, as conversely done by encoder-decoder networks. Therefore, as the temporal ordering of the presented sentences does matter for inference, the representations of such sentences stored in memory are enriched with an additional temporal encoding term.

To conclude, it is also worth to remark that the memory bank in MemN2Ns is only used for read operations, and is flushed every time a new text-question(s) pair is presented to the network. In a sense, the ability to update the memory content and retain contextual representations from previous sentences is not supported by design; whereas it could seem reasonable in the domain of question answering, as the next text-question(s) input pair is generally uncorrelated from the previous one, this approach does not fully exploit the memory capabilities of a memory network. On the other hand, although storing all the input sentences as separate vectors in the memory bank is perfectly feasible in principle,

using a content-based attention model over all possible sentence representations would be highly demanding in terms of both computational and space requirements. For instance, how would it be possible to use a MemN2N to store and question the entire knowledge base of Wikipedia? It turns out that more sophisticated memory addressing methods, like multiscale notions of attention or hashing, need to be properly evaluated to support this model, and the authors intentionally leave this crucial investigation as future work.

- **Dynamic Memory Networks** (DMN) (Kumar et al., 2016) are a powerful class of memory networks which make use of recurrent neural networks and a soft attention mechanism for modelling an episodic memory and performing content-based memory access respectively. Even though DMNs were proposed as a neural network based framework for question answering, these models are flexible enough to solve sequence tagging tasks, classification problems, sequence-to-sequence and transitive reasoning tasks.

  From a conceptual viewpoint, a DMN first computes a dense representation for all the input sentences and the question under evaluation. Then, similarly to the attention-powered encoder-decoder framework described in the previous subsection, the question representation is used as a query to trigger an iterative attention process that searches the inputs in order to retrieve relevant facts from them. This enables the DMN memory module to reason over the retrieved facts so that it can provide a vector representation of all relevant information to an answer module which generates the answer (Kumar et al., 2016).

  When one says "reason over the retrieved facts", it is assumed that an RNN processes the input sentences (or words, depending on the granularity of the problem) sequentially, based on their importance/relevance with the given question, to construct an episode: it is similar to re-creating or simulating the scene described by the text under analysis. Furthermore, even though this has to be chosen at design-time, the network can create multiple episodes hierarchically before returning a memory vector to the answer module. This type of dynamic memory is inspired by the concept of *episodic memory* which is, similarly to attention, a central topic in neuroscience. According to Schacter, Gilbert, and Wegner (2011), episodic memory is defined as the memory of autobiographical events (e.g. times, places, associated emotions) that can be explicitly stated or conjured. It is the collection of past personal experiences that allow an individual to figuratively travel back in time to remember the event that took place at that particular time and place.

  Since dynamic memory networks make extensive use of RNNs, i.e. the main subject of this chapter, we provide a closer overview of such human-inspired models. As depicted in Figure 3.34, a DMN is composed of four different modules each of which is implemented as a recurrent neural network and it is responsible for a specific intermediate task in the

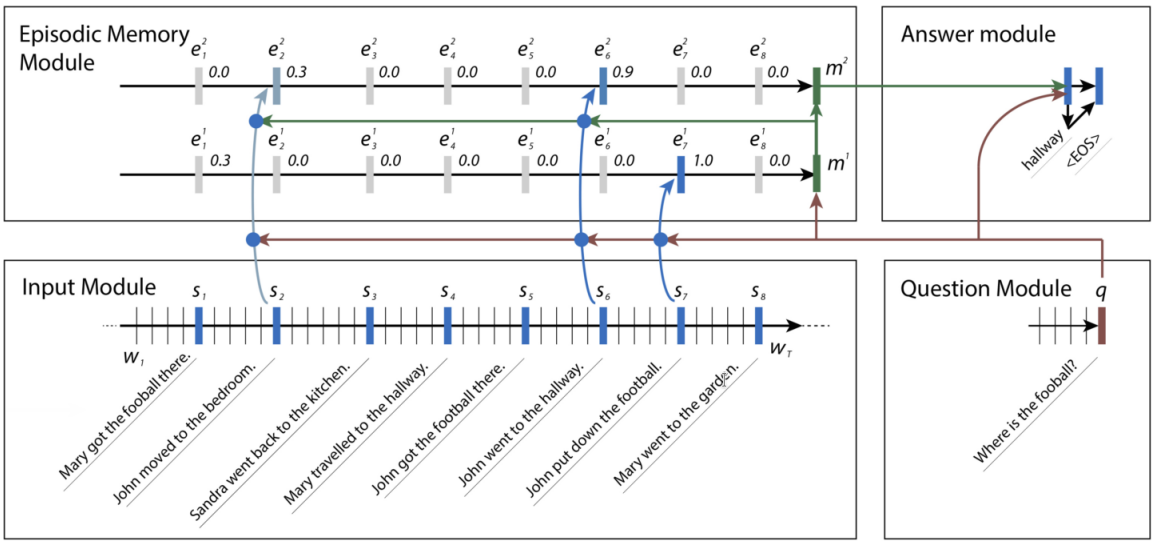


Figure 3.34: Architectural view of a DMN for question answering (as presented by Kumar et al. (2016)) with a real example of an input list of sentences and the attention gates that are triggered by a specific question. As can be seen from the episodic memory module, gate values g_t^i are shown above the corresponding hidden vectors, thus we do not draw connections for gates that are close to zero. More importantly, gates change from one episode to another, supporting the intuition that more passes (episodes) are beneficial for some tasks.

computational flow. More precisely, one can identify the following core components.

1. *Input Module*: encoding the raw text inputs from the task into vector representations and processing the resulting sequence via an RNN. More precisely, a sequence of T_I words $w_1, \ldots, w_{T_I}$ is given as input to an RNN s.t. it can update its hidden state as $h_t = RNN(L[w_t], h_{t-1})$, where L is the embedding matrix and w_t is the word index of the t-th word of the input sequence. Depending on the form of the input text provided by the task, two different scenarios are possible:
   - if the input sequence is a single sentence, the input module outputs all the T_I hidden states of the RNN (like the attention-powered encoder seen before);
   - if the input is a list of sentences, we concatenate them into a sequence of words inserting end-of-sentence (EOS) tokens as delimiters. The hidden states at each EOS token are the final representations of the module.

   For convenience, we denote the output of this module as the sequence of T_C fact representations c, whereby c_t denotes the t-th element in the output sequence.
2. *Question Module*: converting a given question into a distributed vector representation and encoding it via an RNN. In particular, a question made of T_Q words is processed sequentially so that the hidden states for the question encoder at time t is

given by $q_t = GRU(L[w_t^Q], q_{t-1})$, where w_t^Q denotes the word index of the t-th word in the question. Like a traditional encoder (Section 3.1.1), the question module returns a single vector i.e. the final hidden state of the encoder: $q = q_{T_Q}$. The encoded question is then fed into the episodic memory module, and forms the basis, or initial state, upon which the episodic memory module iterates (Kumar et al., 2016).

3. *Episodic Memory Module*: composed of an internal memory, a content-based attention mechanism and a recurrent network to update the internal memory. Considering the episodic nature of the memory recreated in this module, during an iteration i, we focus on the representations c while taking into consideration the question representation q and the previous memory m^{i-1}. By doing so, the context returned by the attention model is used to influence (as gating mechanism) the sequential pass of the fact representations via a recurrent network. Once all the input facts are passed, the RNN manages to produce an episode e^i which is consumed, together with the previous memory vector m^{i-1}, to update the episodic memory as $m^i = GRU(e^i, m^{i-1})$.

   To gain more insights on this intricate process, we need to describe how the attention model is defined, how we precisely exploit the output of the attention model (the context) in the module, as well as the episode generation process.

   Attention mechanism implemented according to the attention model described in Section 3.5.2, but using a gating mechanism as attention function: for each pass i, we take as input a candidate fact c_t, the previous memory m^{i-1}, and question q to compute a gate $g_t^i = G(c_t, m^{i-1}, q)$, where G implements a content-based attention function. However, the novelty of this approach is not in the attention model itself, but in the way the context vector is used.

   Episode generation is the core of the episodic module, as a recurrent neural network is expected to process sequentially the fact representations $c_1, \ldots, c_{T_C}$ and update its hidden state according to the attention of each fact g_t^i. More precisely, to compute the episode for pass i, the authors (Kumar et al., 2016) use a modified GRU over the sequence of the facts weighted by the attentional gates, and the episode vector that is given to the answer module is the final state of the GRU. The equations to update the hidden states of the GRU at time t and compute the episode are given respectively as follows:

$$h_t^i = g_t^i \, GRU(c_t, h_{i-1}^t) + (1 - g_t) h_{i-1}^t \qquad \text{and} \qquad e^i = h_{T_C}^i \tag{3.93}$$

   As demonstrated by Kumar et al. (2016), letting the episodic memory module take multiple passes over the input before returning a memory vector is beneficial for

some tasks. Each iteration provides the module with newly relevant information about the input: the module can thus retrieve new information, in the form of input representations, which were thought to be irrelevant in previous iterations. In so doing, after T_M passes, the final memory m_{T_M} is given to the answer module.

4. *Answer Module*: generating an answer given the final memory vector of the memory module and the encoded question. Depending on the requirements of the learning task, this module is either triggered at each time step or once at the end of the episodic memory (like a traditional decoder). To implement the decoder, the DMN uses a GRU whose initial state z_0 is initialised to the episodic memory output m_{T_M}. At each time step t, it takes as input the last hidden state z_{t-1}, as well as the question q and the previously predicted output y_{t-1} which are concatenated together, to compute the next word of the answer as follows:

$$z_t = GRU([y_{t-1}, q], z_{t-1}) \qquad \text{and} \qquad y_t = softmax(Wz_t) \tag{3.94}$$

Finally, the output can be trained with the cross-entropy error classification of the correct sequence, appended with a special EOS token to halt the decoder.

The dynamic memory network model is using recurrent networks to implement each module confer more flexibility to the proposed approach, in the sense that it can potentially handle different kind of input data and learn several tasks. In fact, DMNs adopt neural sequence models for input representation and processing, attention mechanisms, and response generation, thereby capturing positional and temporal features in a natural manner. This is the reason why the DMN model has been tested on tasks other than QA, and can thus be considered as a flexible architecture for a variety of NLP applications, including classification, question answering and sequence modelling in general.

The same consideration cannot be said for MemNN and MemN2N described before, because they process sentences independently (not sequentially) and this requires a separate feature accounting for the temporal ordering of words or sentences (the positional encoding). Therefore, current implementations highlight that MemNNs and MemN2Ns do not offer a general framework for the broad family on NLP tasks.

However, as similarly observed for the previous memory networks, a full memorisation of the input facts, together with the use of a content-based attention mechanism over all of them, is challenging in practical applications. In this context, a different formulation of the input module under the form of working memory (Graves et al., 2014) is said to be more general to represent knowledge of the state of affairs. In fact, even the authors

(Kumar et al., 2016) raise the problem of scaling the model with larger inputs; however, they propose another direction as future work, i.e. running an information retrieval system to filter the most relevant inputs before running the DMN, or using a hierarchical attention module. A final critical observation regards the number of episodes m_{T_M} to generate in the episodic memory module. As can be noticed from the model definition, this number is chosen at design-time; this is actually similar to the feed-forward depth of MemN2N, which imposes a fixed number of memory accesses (or hops).

- **Neural Turing Machines** (NTM) (Graves et al., 2014) are probably the most direct attempt to equip a recurrent network with an external memory bank for algorithmic tasks, i.e. a category of problems that traditional ANNs typically struggle to solve. This extension corresponds to a potentially large addressable memory which is intended to play the role of the theoretically infinite memory tape of Turing machines. Unlike a Turing machine, an NTM can be considered as a differentiable computer that can be trained by gradient descent, thereby learning programs with output supervision only.

  Whereas a DMN is intended to model the concept of episodic memory, NTMs models the working memory, which has been ascribed to the functioning of a system composed of the prefrontal cortex and basal ganglia (Goldman-Rakic, 1995). Psychologists have extensively studied the capacity limitations of working memory, which is often quantified by the number of "chunks" of information that can be readily recalled (Miller, 1956), but the mechanisms of working memory still remain obscure. However, the verbal definition is understood to mean a capacity for short-term storage of information and its rule-based manipulation (Baddeley, 1992). In computational terms, these rules are *simple programs*, and the stored information constitutes the *arguments* of these programs (Graves et al., 2014). Thus, an NTM resembles a working memory system, as it is designed to solve tasks that require the application of approximate rules to "rapidly-created variables."

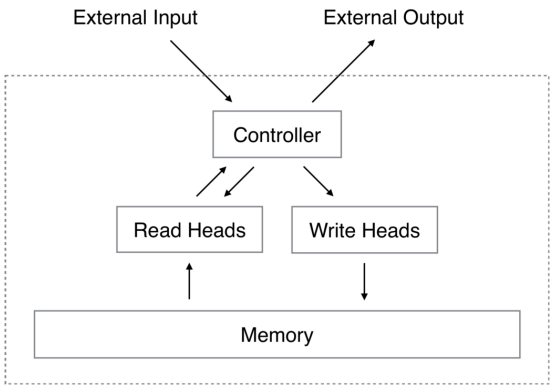


Figure 3.35: High-level diagram of the NTM architecture from (Graves et al., 2014).

As depicted in Figure 3.35, the general architecture of an NTM comprises two core components: a neural network *controller* and a *memory bank*. Like most neural networks, the controller interacts with the external world via input and output vectors. Unlike a standard network, it also interacts with a memory matrix using selective *read* and *write* operations called "heads". To make every component differentiable, a soft-attention mechanism overcomes the discrete nature of traditional memory access and allows the recurrent network to learn where to read and write every time a new input in the sequence is processed. More precisely, at each step, we *read from* and we *write to* every location to different extents according to an attention distribution.

To formally describe the read and write heads, let M_t be the contents of the $N \times M$ memory matrix at time t, where N is the number of memory locations, and M is the vector size at each location. The read and write heads can thus be defined as follows.

- A **read head** with a key vector k_t is first passed to an attention model which computes an attention distribution α_t over the N locations at time t (the details of how the attention distribution is computed are given in moment). Therefore, the length M vector r_t returned by the read head is defined as a convex combination of the row-vectors $M_t(i)$ in memory as follows:

$$r_t = \sum_i \alpha_t(i) \odot M_t(i) \tag{3.95}$$

  Instead of specifying a single location, the attention distribution describes how we spread out the amount of importance about different memory positions (Olah & Carter, 2016). In other words, we can think of this process as reading from every location to the extent given by the attention distribution.

- A **write head** is performed taking inspiration from the input and forget gates in LSTM, thus it involves an *erase* followed by an *add* operation. Given an attention distribution α_t returned by the attention mechanism for writing a key vector k_t at time t, the memory vectors $M_{t-1}(i)$ from previous time-step are updated as:

$$M_t(i) = \alpha_t(i)k_t + (1 - \alpha_t(i))M_{t-1}(i) \tag{3.96}$$

  Analogously to the read head, this operation is similar to writing everywhere at once to different extents according to the given attention distribution (how much we write at every location). In fact, the new value of a position in memory is a convex combination of the old memory content and the write value, with the position between the two decided by the attention weight (Olah & Carter, 2016).

Now that both the read and write heads are presented, we can outline how the attention distribution α_t is produced at each time step. The attention model used in the NTM is more complex than the basic one introduced at the preamble of this section, as it makes joint use of two attention mechanisms for addressing memory content.

1. The first addressing mechanism is based on a **content-based** attention model that focuses attention on locations based on the similarity between their current values and values emitted by the controller. It is the same attention mechanism used by the decoder-decoder models described in Section 3.5.2 as well as by the memory networks introduced so far. The advantage is that retrieval is simple, as it requires the controller to produce an approximation to a part of the stored data, which is then compared to memory to yield the exact stored value (Graves et al., 2014).

   More precisely, the (write or read) head first produces a length M key vector k_t that is compared to each vector $M_t(i)$ by a similarity measure $K[\cdot,\cdot]$ (e.g. cosine similarity). The system produces a normalised weighting α_t^c based on the similarity and a positive key strength, β_t, which can amplify/attenuate the precision of the focus. The content-based addressing is thus computed as follows:

   $$\alpha_t^c(i) = \frac{\exp(\beta_t K(k_t, M_t(i)))}{\sum_j \exp(\beta_t K(k_t, M_t(j)))} \tag{3.97}$$

   Then, before returning the so computed attention distribution, the head emits a scalar interpolation gate g_t in the range $(0,1)$. This value is used to blend between the weighting α_{t-1} produced at the previous time step and the weighting α_t^c produced by the content system at the current time step as follows:

   $$\alpha_t^g(i) = g_t \alpha_t^c(i) + (1 - g_t)\alpha_{t-1}(i) \tag{3.98}$$

   Therefore, a gate $g_t = 0$ would use the weighting from the previous time step and ignore the content-based attention distribution; if equal to one, the weighting from the previous iteration is ignored, and content-based addressing is considered.

2. As not all problems are well-suited to content-based addressing, we also need to use a **location-based** attention mechanism for memory addressing. In fact, in arithmetic tasks the content of a variable is arbitrary, but the variable still needs a recognisable name or address. For instance, if we need to multiply two variables x and y, a controller for this task could take the values of the variables x and y, store them in different addresses, then retrieve them and perform multiplication. Thus, variables are addressed by location, not by content.

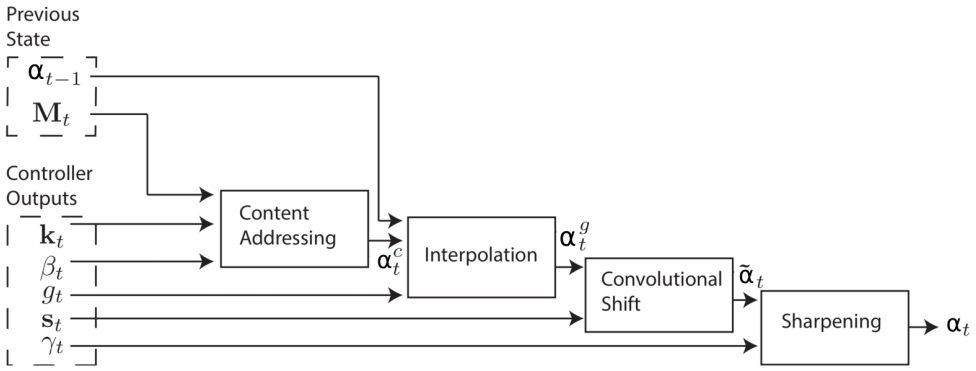


Figure 3.36: The attention model used in NTM from (Graves et al., 2014). A key vector k_t and key strength β_t, are used to perform content-based addressing of the memory matrix. The resulting weighting is interpolated with that from the previous time step via the interpolation gate g_t. The shift weighting s_t determines whether and by how much the weighting is rotated. Finally, the weighting is sharpened via γ_t and used for memory access.

The location-based addressing mechanism makes it possible to iterate across the memory locations and perform random-access jumps by implementing a rotational shift of the weighting. For instance, a rotation of 1 would shift the focus to the next location if the current weighting focuses entirely on a single memory position (Graves et al., 2014). To implement this idea, each head emits a shift weighting s_t defining a normalised (thus differentiable) distribution over the allowed integer shifts. Assuming to index the N memory locations from 0 to $N-1$, the rotation applied to α_t^g by s_t can be expressed as a circular convolution:

$$\tilde{\alpha}_t(i) = \sum_{j=0}^{N-1} \alpha_t^g(j) s_t(i-j) \tag{3.99}$$

where all index arithmetic is computed modulo N. As the convolution can cause dispersion of weightings over time if the shift weighting is not sharp, each head emits one further scalar $\gamma_t \geq 1$ whose effect is to sharpen the final weighting:

$$\alpha_t(i) = \frac{\tilde{\alpha}_t(i)^{\gamma_t}}{\sum_j \tilde{\alpha}_t(j)^{\gamma_t}} \tag{3.100}$$

This powerful addressing system can thus operate in three modes: (i) a weighting can be chosen by the content system without any influence by the location system; (ii) a content-based weighting can be chosen and then shifted (this allows to find a contiguous block of data, then access a particular element); (iii) a weighting from the previous time step can be rotated with no influence from the content-based system (we can iterate through a sequence of addresses by constantly advancing at each time-step).

The NTM architecture has several free parameters to design, including the size of the memory and the type of neural network used as the **controller**. Using a recurrent controller such as LSTM, which already maintains its own internal memory, can complement the memory bank and allow the controller to mix information across multiple time steps of operation. In this case, it is compelling to compare the controller to the central processing unit in a computer and the memory matrix to RAM, so that the hidden states of the recurrent controller are akin to the registers in the processor (Graves et al., 2014).

From a critical point of view, and to our best, an NTM is the only memory network model which makes use of a working memory as we would generally expect: we can retrieve facts according to content and location, and it is possible to update the memory block in order to *accumulate* new factual knowledge. However, this architecture was specifically designed to solve algorithmic tasks such as sorting, copying and recall, and it is not directly applicable to other tasks unless a more generalised model is derived.

	Supervision	Sequential processing	Attention type	Multiple passes	Learning tasks
MemNN	strong	no (time injection)	hard (content-based)	yes (fixed)	QA
MemN2N	weak	no (time injection)	soft (content-based)	yes (fixed)	QA
DMN	weak	yes	soft (content-based)	more episodes (fixed)	QA + NLP
NTM	weak	yes	soft (content and location based)	?	algorithmic

Table 3.2: Comparison of the outlined attention-powered memory neural networks.

As several different criteria can be used to compare memory networks, we outline each of them (Table 3.2) to highlight both the limitations and strengths of each contribution to support the narrative of our discussion.

- *Supervision and attention*: whereas the implementation of MemNN proposed in (Weston et al., 2014) makes use of a hard-attention model for memory addressing and need strong supervision in terms of which specific factual information has to be retrieved for a given query (i.e. a question vector), all the other memory network models use soft-attention mechanisms and do not need such a strong kind of supervision. As a result, Weston's model of MemNN can only be used for those QA problems in which a label of the most relevant sentences for each text-question input pair is available.

  Moreover, a closer look at the specific attention models discussed in this section reveals that only NTMs use a sophisticated chain of operations (Figure 3.36) for the purpose of a more flexible and granular memory access. This mechanism allows the controller (i.e.

the recurrent network exploiting the memory bank) to retrieve factual knowledge from memory based on both content and location, as well as to perform loops and random access whenever an algorithmic procedure is triggered. On the other hand, MemNN, MemN2N and DMN only make use of a content-based attention model to retrieve relevant facts from memory (read operations); in addition, their memory block acts as a sort of unbounded space: a new fact is just stored in a free location without affecting the content of the non-empty locations (write operations are simple accumulators).

- *Sequential processing*: as previously mentioned, both the implementations of MemNN and MemN2N do not use recurrent networks to process sequential data, nor to perform content-based memory access. In fact, these architectures are completely feed-forward and the resulting models can only process sentences independently. Therefore, they explicitly require bag of n-gram vector features together with a separate feature capturing the temporal order of sentence (i.e. positional encoding). As a consequence, MemNNs and MemN2Ns are rather limited in their applicability and need major architectural changes if one aims to use them for tasks other than QA.

  In contrast, NTMs and DMNs process input data sequentially and make extensive use of recurrent neural networks to implement the different components of their architectures. This crucial consideration make them directly applicable to a broader range of applications without significant architectural adjustments and feature engineering.

- *Memory accesses*: among the strengths of memory networks in general, we find the possibility to perform multiple memory hops with a query vector before inferring the output. Each memory access provides the network with newly relevant information about the input. This corresponds to the ability to progressively retrieve new information which was thought to be irrelevant during the previous memory hops. As can be seen from Table 3.2, both MemNN, MemN2N and DMN implement such a mechanism, but they constrain the network to perform a fixed number of memory hops. Therefore, the modeller has to decide at design-time how many memory accesses the network has to do before outputting a symbol; this number can thus be restrictive in case the current context requires more reasoning over the memory bank, or even excessive if no more facts need to be retrieved.

  Conversely, it is not clear how many memory hops an NTM is supposed to perform once presented with a certain input sequence. The ability of the controller to emit read and write heads independently gives us the impression that the number of memory hops may somehow depend on the context. However, this is not clearly explained by Graves et al. (2014) thus a closer investigation is needed to clarify this point.

- *Learning tasks*: whereas the proposed implementation of MemNN, MemN2N and DMN were addressed to question answering, the NTM model has been designed for solving purely algorithmic tasks such as sorting, copying and recall. In sum, every model seems particularly specialised in a specific family of problems.

  Nevertheless, to advocate dynamic memory networks, an effective model for QA might be used for other natural language processing tasks if based on sequential processing. In fact, as question answering requires an understanding of the meaning of a text and the ability to reason over relevant facts, most NLP tasks can be cast as a QA problem. As mentioned in (Kumar et al., 2016), we can consider the following tasks:

  - high level tasks like machine translation (what is the translation into French?);
  - named entity recognition (what are the named entity tags in this sentence?);
  - classification problems like sentiment analysis (what is the sentiment?);
  - part-of-speech tagging (what are the part-of-speech tags?).

  However, there still is no evidence as to whether a memory network for QA could also be an effective model for other natural language processing problems.

It is worth remarking that both DMN and NTM can boast a neuroscientific interpretation, as they intend to model working and episodic memory respectively. This suggests that seeking inspirations from the field of neuroscience can provide valuable insights to design new powerful memory neural networks. An investigation on how to combine the strengths of the models presented here, together with the integration of novel and more sophisticated attention mechanisms for memory addressing and manipulation, would possibly be an interesting path to follow.

Adaptive computation time

As defined in Section 3.1, RNNs perform the same amount of computation for each time step, i.e. 1 hidden state update per step, resulting in $O(n)$ operations for a sequence of length n. However, for both computational efficiency and ease of learning, it seems preferable to dynamically vary the number of steps for which the network "ponders" each input before emitting an output: this is what Olah and Carter (2016) describe as "*thinking more when things are hard*". This requirement seems to be particularly relevant for synthetic tasks like determining the parity of binary vectors, applying binary logic operations, adding integers and sorting real numbers.

For that aim, another way to exploit attention is to let a recurrent network learn how many computational steps to take between receiving an input and emitting an output. The corresponding algorithm goes under the name of Adaptive Computation Time (ACT) (Graves, 2016) and it

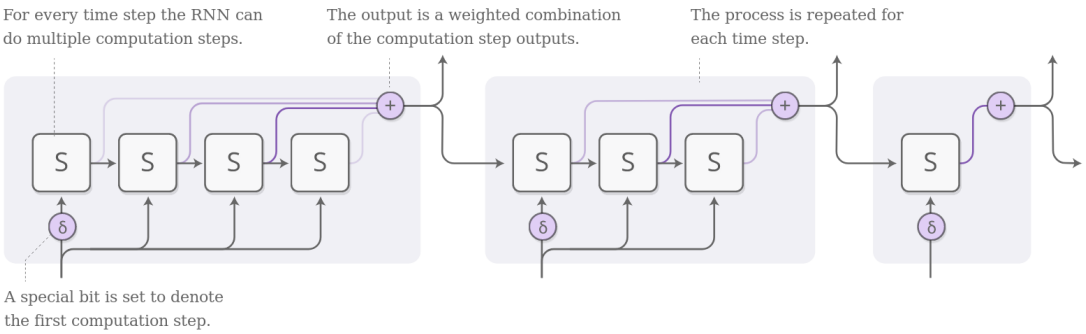


Figure 3.37: Example of adaptive computation time in RNN, from Olah and Carter (2016).

requires minimal changes to the network architecture other than being deterministic and differentiable. Similarly, learning the number of computational steps to perform at a certain time step is achieved by an attention-based model; we can think of ACT as modelling an attention distribution over the number of computational steps to run for each time step, s.t. the output at time t is a weighted combination of the outputs of all the computational steps taken (Figure 3.37).

On a conceptual level, ACT augments the network with a sigmoidal halting unit whose activation determines the probability that computation should continue. The resulting halting distribution is used to define a mean-field vector for both the network output and the internal network state propagated along the sequence (Graves, 2016). Such deterministic approach has the advantage of using a smooth function of the outputs and states, with no need for stochastic gradient estimates. Moreover, to achieve a parsimonious use of computation and encourage faster solutions, a regularisation term called time cost is added to the loss function. Therefore, the RNN has to learn to trade off accuracy against speed.

To provide a formal description of ACT, we recall the basic definition of an RNN composed of a matrix of input weights W_x, a parametric state transition model $\mathcal{S}$ depending on the type of recurrent unit, a set of output weights W_y and an output bias b_y. While processing an input sequence $x = (x_1, \ldots, x_T)$, the RNN computes the state and output sequences $s = (s_1, \ldots, s_T)$, $y = (y_1, \ldots, y_T)$ by iterating these equations from $t = 1$ to T:

$$s_t = \mathcal{S}(s_{t-1}, W_x x_t) \tag{3.101}$$

$$y_t = W_y s_t + b_y \tag{3.102}$$

To perform a variable number of state transitions and compute a variable number of outputs at each input step, ACT proposes a few architectural extensions and adaptations. Assuming to perform $N(t)$ updates at step t (instead of 1), we can define the *intermediate state sequence* as $(s_t^1, \ldots, s_t^{N(T)})$ as well as the *intermediate output sequence* $(y_t^1, \ldots, y_t^{N(T)})$ at step t as:

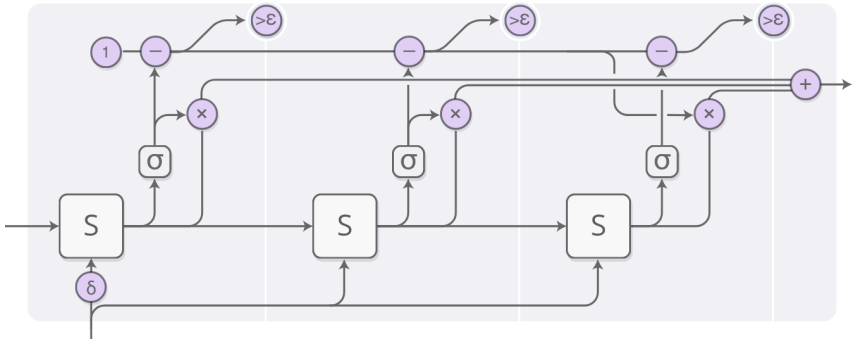


Figure 3.38: Simplified architectural view of an ACT-powered RNN at a single time step (from (Olah & Carter, 2016)). In this example, the number of computational steps (or state updates) performed is equal to 3.

$$s_t^n = \begin{cases} \mathcal{S}(s_{t-1}, x_t^1) & \text{if } n = 1 \\ \mathcal{S}(s_{t-1}^{n-1}, x_t^n) & \text{otherwise} \end{cases} \tag{3.103}$$

$$y_t^n = W_y s_t^n + b_y \tag{3.104}$$

where $x_t^n = x_t + \delta_{n,1}$ denotes the input at time t, augmented with a binary flag that indicates whether the input step has just been incremented; this allows the RNN to distinguish between repeated inputs and repeated computations for the same input. Next, to determine how many updates the network performs at each input step:

1. an extra sigmoidal *halting unit* h is added to the recurrent output, with associated weight matrix W_h and bias b_h (shared across computational steps and times) following

$$h_t^n = \sigma(W_h s_t^n + b_h); \tag{3.105}$$

2. the activation of the halting unit is then used to compute the *halting probability* p_t^n of the intermediate steps, i.e. the probability to stop at computational step n as

$$p_t^n = \begin{cases} R(t) & \text{if } n = N(t) \\ h_t^n & \text{otherwise} \end{cases} \tag{3.106}$$

where the second case indicates that the output of the sigmoid neuron is directly used as halting probability, and the first case is just a remainder mechanism (i.e. $R(t)$) to ensure that all probabilities sum to one once the last update is performed. As better described by the following equations, it is like having a total budget for the halting weights of 1

(at each time step). In so doing, when we stop after $N(t)$ updates, the remaining halting budget is attributed to the last computational step. More formally, we define the remainder mechanism $R(t)$ and the number of computational steps/updates $N(t)$ per time step t as:

$$N(t) = \min(n' : \sum_{n=1}^{n'} h_t^n \geq 1 - \varepsilon) \tag{3.107}$$

$$R(t) = 1 - \sum_{n=1}^{N(t)-1} h_t^n \tag{3.108}$$

where ε is a small constant (e.g. 0.01) whose purpose is to allow computation to halt after a single update if $h_t^1 \geq 1 - \varepsilon$; otherwise, a minimum of two updates would always be required for every input step.

At this point, instead of proceeding stochastically, Graves (2016) suggests using p_t^n to determine mean-field updates for the states and outputs as indicated by the following equations:

$$s_t = \sum_{n=1}^{N(t)} p_t^n s_t^n \qquad y_t = \sum_{n=1}^{N(t)} p_t^n y_t^n \tag{3.109}$$

However, this approach relies on the assumption that the states and outputs are approximately linear: we are implicitly expecting that a linear interpolation between a pair of state or output vectors would also interpolate between the properties the vectors embody. Nevertheless, Graves (Graves, 2016) identifies different arguments supporting this assumption; moreover, empirical results highlight the tendency for both mean-field and stochastic latent variables to concentrate all the probability mass on a single value as training converges.

The last modification we need to make to implement ACT in our RNN consists in introducing a regularisation term to limit the amount of computation the network performs. In fact, if no constraints are placed on the number of updates we can take per step, the RNN would be eager in terms of "pondering" each input for as long as possible. Hence, given a length T input sequence x, we define the ponder sequence $(\rho_1, \ldots, \rho_T)$ and the ponder cost as follows:

$$\rho_t = N(t) + R(t) \tag{3.110}$$

$$\mathcal{P}(x) = \sum_{t}^{T} \rho_t \tag{3.111}$$

Considering that we aim at minimising the total number of updates (or computational steps) performed by the RNN (i.e. the sum of all $N(t)$), which is not differentiable, the ponder cost

$\mathcal{P}(x)$ offers an upper bound of that information (since $R(t) \in (0,1)$). Therefore, we can encourage the network to be computationally parsimonious by introducing $\mathcal{P}(x)$ to the sequence loss function used for training as follows:

$$\hat{\mathcal{L}}(x,y) = \mathcal{L}(x,y) + \tau\mathcal{P}(x) \tag{3.112}$$

where τ is a hyper-parameter called *time penalty* that weights the relative cost of computation versus error. However, the experiments proposed by Graves (2016) demonstrated that the behaviour of the RNN is quite sensitive to the value of this hyper-parameter, and choosing a good value for it would require having a quantitative measure of both computation time and prediction error (e.g. if the relative financial cost of both were known). This represents one of the most crucial drawbacks of ACT, and it is probably attributable to the fact that the time penalty regularisation term could be better engineered.

Another critical observation regards the possible benefits that ACT can bring to a recurrent neural network in terms of improved performance on sequential learning tasks. As observed by Graves (2016) from his experiments, the advantages of extra computation largely depend on the specific type of learning problem. More precisely, whereas ACT was demonstrated to dramatically improve the performance of an RNN on synthetic tasks like determining the parity of binary vectors, applying binary logic operations, adding integers and sorting real numbers, there is no considerable gain in language modelling problems. Therefore, we can probably conclude that not all tasks require extra computation as well as the availability of a large addressable memory bank for storing factual knowledge.

Nevertheless, the use of ACT is quite promising at different stacked recurrent layers and together with other soft attention modules, which it could enable to dynamically adapt the number of glances or internal operations they perform at each time-step (Olah & Carter, 2016). For instance, in the context of memory networks, it would be interesting to investigate how this approach can be exploited to perform a variable number of memory hops (according to the context), instead of forcing the network to access its memory the same number of times. This would allow a neural network to overcome its architectural limitations and confer a more flexible and dynamic use of memory when it comes to dealing with factual knowledge.

Neural programmers

As pointed out when we introduced NTMs, traditional deep network models are capable of learning the fuzzy underlying patterns in the data, but they have not had similar impact in applications that involve crisp reasoning. Even recurrent neural networks with attentional interfaces (Section 3.5.2) struggle to perform arithmetic and logic tasks (Neelakantan et al., 2015),

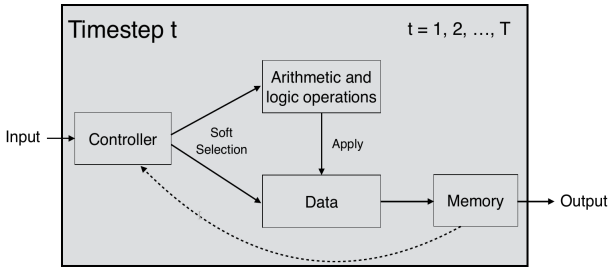


Figure 3.39: Basic architecture of a Neural Programmer from (Neelakantan et al., 2015). At each time step, the controller selects both the operation to perform (from the instruction set) and the data segment on which to operate (from the memory bank) via two attention mechanisms. Another memory stores the output of the previous operations applied to the segments. The controller runs for several steps thereby inducing compositional programs.

whereas memory network can provide one of the missing ingredients needed to accurately solve these problems. Indeed, in addition to an addressable memory with attention-based access, we also need a similar approach to augment an RNN with a set of operations.

This is the intuition behind a neural programmer (Neelakantan et al., 2015), i.e. a recurrent architecture exploiting a memory bank and a predefined set of operations so that it can learn to induce programs by selecting and combining operations in order to perform a certain task. More precisely, as illustrated in Figure 3.39, a neural network is augmented with a separate memory block (a sort of knowledge-base related to the problem) and small set of basic arithmetic and logic operations (called *instruction set*) that can be trained end-to-end using back-propagation. To make program induction possible, the architecture is made of different of components which are orchestrated by an RNN controller. As the program is generated one operation at a time by the controller, we need to make sure that several steps are run as follows,

1. At each time step, the controller selects a data segment from the memory block and a particular operation from the instruction set to apply to that segment. Analogously to memory networks, the selection process for the data source and operations is performed by two soft-attention models respectively, and it is supported by a history RNN; as the name suggests, the purpose of this last component is to remember the previous operations and data segments selected by the model till the current time step.

   As a result of the soft-attention mechanisms, the resulting approach is differentiable end-to-end and the whole neural network can be trained jointly by gradient descent. As can be seen in Figure 3.40, the controller uses an attention model to obtain a probability distribution over the possible operations, so that it can evaluate what the next operation should be. Therefore, instead of running a single operation at each step, we let the network perform all of them and then average the outputs together, weighted by the attention

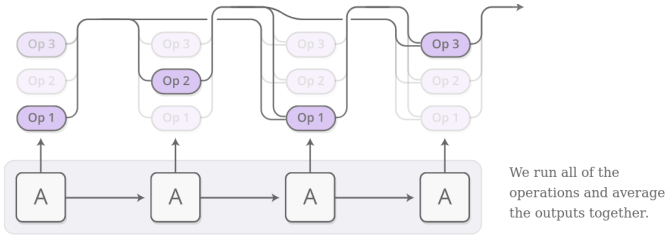


Figure 3.40: Program induction with soft-attention on operations, from (Olah & Carter, 2016).

probability we ran that operation. In the same way as we saw for memory networks (Section 3.5.2), an attention model regulates the retrieval of the relevant data segments on which the operations will be applied.

2. The network then propagates the output of the averaged operations forward at every step, so that it can form the final, more elaborated output. The results of these operations are also passed to the history recurrent neural network so that we can keep track of the previous stages/states of the program under induction.

3. Using just the target output of the program, we can adjust the network to select the right data segments and operations, thereby inducing the correct program.

At test time, the soft-attention based selection of data segments and operations is replaced with hard selection. As declared by the authors (Neelakantan et al., 2015), the neural programmer is run for a total of T time steps chosen in advance to induce compositional programs of up to T operations. Therefore, as observed for other attention-powered recurrent architectures presented in this section, the model is constrained to perform a fixed amount of computation and a single memory hop per step in order to induce a program for a certain task.

It is worth mentioning that a neural programmer has two attractive properties.

- It learns to generate programs from a weak supervision signal, which is the result of execution of the correct program. Thus, it does not require the expensive annotation of the correct program for the training examples.

- It does not require additional rules to guide the program search. Indeed, the algorithmic designer only defines a list of basic operations, which should require lesser human effort than in previous program induction techniques.

The resulting approach can thus be considered as a general framework for neural program induction. The neural programmer model has indeed demonstrated that by combining recurrent

networks with mathematical operations, we can utilise both the fuzzy pattern matching capabilities of deep networks and the crisp algorithmic power of digital computers so as to solve tasks which have not been previously attempted by neural networks.

As the authors (Neelakantan et al., 2015) proposed an implementation of the neural programmer for the specific problem of generating SQL-like programs to answer questions about tables, we are not going to discuss their model. However, we believe that one of the limitations of their approach is to constrain the controller to perform no more than T operations per program. Furthermore, not only does a neural programmer has to learn how to induce a program for a certain task (in terms of a sequence of operations performed on specific data segments), but also to learn how to actually execute the resulting program operation by operation. An alternative and intuitive approach would be to use a neural programmer to generate a program (only for the creative process) whereas the execution of each operation is left to a digital computer.

Chapter 4

Predictive models for music

This chapter provides an in-depth overview of the music modelling problem, starting from the properties and the main technical challenges involved (Section 4.1). Following this introduction, Section 4.2 categorises the most relevant works in the literature according to the methodology proposed for addressing one, or more, technical challenges. This enables a systematic analysis of the literature, revealing how these technical challenges are still not fully addressed by the current works (Section 4.3).

4.1 Music modelling: properties and challenges

In this part, we present the main properties characterising the music modelling problem, together with the major technical challenges which are inherently related to the complexity of the music signal. The technical challenges are presented at a taxonomical level, i.e. from *monophonic* and *polyphonic melody modelling* to *multi-track music modelling*, thus emphasising the increasing difficulty of these tasks as more musical aspects need to be taken into account.

4.1.1 Properties of the problem

As introduced in Section 2.1, music modelling is a specific instance of the more general problem of sequence modelling – a prediction task, where the intrinsic nature of the data makes it difficult to capture all those complex and latent aspects characterising the observed phenomenon. As a consequence, achieving accurate predictions depends on a large number of factors. Whereas some of these factors cannot be identified a priori, music has always been a subject of study from several different fields, from psychology and neuroscience to music theory and music informatics, thus suggesting that a great deal of knowledge could potentially be exploited, in different ways and forms, to simplify the modelling problem.

From a computational perspective, the most common properties of music modelling, that are fundamental to understand and address the corresponding task, can be summarised as follows: (i) how to represent musical sequence; (ii) modelling and learning musical sequences; and (iii) the evaluation strategies for both the quantitative and qualitative assessment of the model.

Representing musical sequences

One of the most crucial design choice is about choosing or designing a proper encoding of the musical sequences under study. Not only the adopted encodings will affect other modelling choices, such as the architecture of the model itself (e.g. the input and output layers of an artificial neural network), but also the musical features learned by the model during training.

As we are dealing with symbolic music, and considering that alternative symbolic representations (c.f. Section 2.1.3) can be converted into one another without loss of information, we can visualise a musical sequence as a piece transcribed in a music sheet, where notes flow temporally on the pentagram from left to right following music notation rules. A proper representation of those musical sequences should be able to address the following aspects: (i) how to represent the *evolution of time* in a piece; (ii) how to represent each note appearing in a sequence with regards to both its main features, i.e., *pitch* and *duration*.

Time representation Since music is a temporal process, the representation of time is fundamental for music modelling, and it is generally regarded as the first design choice to make. For this purpose, let us consider the following general methodologies:

Global time. In the extreme case, one might think of feeding a model with the entire musical sequence in a single pass, in the form of a static input (as usually done for images). As can be observed, this encoding does not require temporal processing, and a traditional (static) feed-forward neural architecture would be suitable for such a task, as long as a masking mechanism is adopted in order to prevent the model from accessing future tokens.

Windowed-time. As a middle ground, and requiring fewer input units than the previous encoding, we could also consider a (fixed-size) sliding window of successive time steps. More precisely, the time-as-spatial-position representation converts the problem of learning musical sequences into the problem of learning spatial patterns. If the window size is chosen to match a single measure (or bar), learning a musical sequence would thus correspond to learning to associate each measure of a piece with the next measure.

Note step. Treating music as a purely sequential phenomenon, with notes being processed one after another, temporality is implicitly encoded in the relative position of a note in the

sequence (rather than its spatial position in a window of units). In contrast to the previous representations, this type of encoding is more suitable to a sequence model, which learns to predict a sequence of single notes rather than a set of notes simultaneously. Hence, the most granular token to be processed corresponds to a note, regardless of its duration.

Time step (or *time slice*). The atomic temporal granularity of the input is a local temporal slice of the musical content, corresponding to a specific temporal moment (time step). In particular, a time step can be expressed as a frame (e.g. 10ms) or in terms of metrical duration (e.g. 16th note). The process of dividing a musical sequence into discrete time steps is called *quantisation*. The first approach is commonly referred to as *frame-level quantization*, whereas the second one does not adhere to a specific naming convention, but depends on the chosen metrical reference; for instance, if we consider the 16th note as the basic unit of quantization, we might refer to a *quantization to the 16th note*.

To the best of our knowledge, the second approach is by far the preferred representation of time in musical sequences, and it is good practise to choose the quantization unit as the shortest note duration (e.g. 16th, 32th or 64th), even though it is usually set larger in order to reduce the length of the resulting (quantised) sequences.

As can be observed, the *windowed-time* and the *step-based* representations are not contradictory among each other, and might be combined together to advantage. In the form of hybrid representation, a sequence model processing time windows (e.g. bars) rather than single notes, would learn a different set of associations and so make different generalisations (Todd, 1989).

More importantly, a corollary of the reported methods is that for both the *global* and *windowed* encoding styles, the representation of a musical object needs to have fixed size (a maximum number of tokens). This is not required if using a step-based approach, where sequences can be of variable length. When designing a predictive model for music, processing sequences of arbitrary length is an important desideratum if a music model is considered for the use cases outlined in Section 2.1.2 (e.g. automatic music transcription). Henceforth, this section focuses on step-based representations, due to the desired flexibility of such encodings.

Pitch representation The next design decision concerns the encoding of notes, and in particular their *pitch* and *duration*. To represent the pitch of a given note, we could either use a unique identifier of that pitch, or the relative transition (i.e. the interval) between that of the previous note in the sequence. More precisely we may consider:

Absolute pitch representations – the actual value of that note according to a reference system. More precisely, there are two main ways of encoding the absolute pitch of a note:

value encoding, intended to represent a single pitch according to a numerical and meaningful value which could potentially capture its audio characteristics, e.g. the corresponding frequency value (in Hertz), or simply the relationship with other surrounding notes, e.g. the corresponding pitch number in MIDI notation, or even a real number in the interval $[0, 1]$ obtained after scale normalisation.

local encoding (alias *one-hot* or *multi-hot* encoding), where the idea is to represent pitches as elements drawn from a pitch vocabulary. The latter is usually chosen to correspond to the pitch range of a grand piano, thus allowing to represent 88 distinct pitches – from A0 to C8. At each time step t, a binary vector $\mathbf{p}^{(t)}$ of size equivalent to the chosen pitch range, is used to represent the active pitches, i.e. those started at t or sustained from previous steps. Each element, or entry, in the binary vector corresponds to a specific pitch (e.g. p_1 for A0, p_{88} for C8). The presence of a specific pitch in $\mathbf{p}^{(t)}$ is denoted with a value of 1 in the corresponding position (e.g. $p_1 = 1$ if A0 is playing), whereas elements associated to 0 are considered off (not playing).

According to Briot et al. (2017), value encoding is rarely used in symbolic music modelling, although it is more common in audio processing due to the risk of loosing accuracy as a consequence of numerical operations, such as approximations. On the other hand, local representations are discretised by nature, hence more robust than the analogue counterparts.

Relative pitch representations. The pitch of a given note is represented in terms of the relative transition (i.e. the interval) between the previous pitch in the sequence. Compared with absolute encodings, the pitch-interval representation is appealing for the following reasons:

- These representations are not bounded in a specific (fixed) pitch range, whereas absolute pitch representations can only encode pitches within the predefined pitch range defined by the vocabulary (at design time).
- Music is encoded in a key-independent fashion, i.e. aside from the initial specification of the starting (absolute) pitch value, the output of the model would not contain any specification regarding the current key (e.g. Cmaj rather than Gmin). As a consequence, the resulting representations would potentially allow the model to learn key-independent representations more easily. As explained by Todd (1989), if the model has to learn two different melodies in two different keys, then using an *actual-pitch* representation might obscure patterns of pitch movement present in both. For example, if one melody included the phrase C-G-E-F and the other included the phrase F#-C#-A#-B (the same sequence transposed 6 half-steps above), the model would be likely to miss any connection between the two, since they use to tally different output

units. In a pitch-interval representation, though, these two phrases would be coded identically, allowing the model to capture the common pattern.

- Key-independence allows transposition of an entire melody simply by changing the initial (absolute) pitch in the sequence, as all the subsequent intervals would remain unchanged regardless of the new key. In contrast, transposing musical sequences in all the keys with an absolute-pitch representation, a method which is often employed for data augmentation, would change every single pitch in those sequences.

Nevertheless, it is worth to observe that the flexibility of interval-based representation is a double-edged sword, and can also be interpreted as a drawback. When an error is made during the prediction of a sequence of intervals, the rest of the sequence would be completely transposed to a different key – it would diverge without returning to the original key. Todd (1989) observes that the advantages of using interval representations could still be retained, and their drawbacks minimised, by incorporating them into a hybrid representation. Intervals could be used for the majority of the model's outputs, as long as absolute pitches are calibrated every so often. For instance, at the start of every measure (or other suitable period), the actual pitch of the melody could be provided to the model. This would re-calibrate the musical sequence periodically, so that it remains in the proper key.

To our best, most works in the literature adopt absolute-pitch representations, and in particular local encodings, due to their ease of use and the robustness in terms of key resilience after mistakenly predicted notes. Moreover, using local representations of pitches makes it possible to formulate the prediction problem as a classification task, where each pitch in the chosen vocabulary corresponds to a distinct class.

Duration representations Representing the duration of notes is another crucial design choice affecting the configuration of both musical sequences and the architecture of the model. Unlike pitch representations, this design choice is strictly related to the representation of time described before. The two most common approaches are presented as follows.

Absolute duration. In case a note-step based representation of time is chosen, the most natural choice is to encode the duration of a note as a separate feature, either in a *local* fashion (e.g. with one unit associated to a quarter-note, another designating an eight note, and so forth); or in a *distributed* manner (e.g. with the number of activated units representing the duration of the note in terms of its 16th constituents). In the most parsimonious settings, at least two distinct features are needed to represent a single note – one for the pitch (e.g. using an absolute value-based encoding) and another one for the duration.

Cumulative duration. One of the most preferred solutions in the literature consists in exploiting the flexible time discretization of musical sequences brought by a quantisation-based representation of time. More precisely, as we divide musical sequences into equally spaced slices of fixed length, each slice automatically encodes the pitches of those notes that are still "active" during that unique time slice. In doing so, the duration of a particular note would correspond to the number of successive pitch vectors in which its constituent pitch(es) stays on. If a frame-level quantization to 10ms is chosen, the duration of a single note would then be expressed as a multiple of that time window. Alternatively, with a metric-level quantization to the 16th note, a note would be represented in terms of 16th multiples; for instance, a C2 quarter note would be represented by the same note repeated (held) for 4 time slices (thus appearing in 4 successive pitch vectors).

Putting all these design dimensions together, if we choose a time-slice representation of time, together with absolute local pitches and cumulative durations for notes, we obtain a *piano-roll* – the most common type of symbolic representations in the field. We can easily think of a piano-roll sequence as expressing a musical sequence in terms of a function of *pitch versus time*, with the model giving the pitch value of this function at equally spaced intervals of time.

Despite their popularity, encodings using *cumulative durations*, such as the piano-roll, cannot directly represent note repetition. This is due to the lack of an explicit way to determine when a note ends. In other words, repeating the same note in two consecutive time steps (a *note articulation*) would not be distinguished from holding that note for the same steps (e.g., a C2 quarter note and four consecutive C2 16th notes would have the same encoding).

Even though most contributions in the literature disregard this limitation, piano-rolls can be extended to address this problem, at the cost of more complex representations. One solution is to introduce a *note begin* event to explicitly mark the start of a note. As advocated by Todd (1989), this binary unit would be activated (1) for those time slices where a new note is played, and off (0) when the time slice covers the continuation of a (previously started) note. Although this solution was initially proposed for monophonic melody modelling – where only a pitch can be active at a certain time step, this approach is not directly applicable to polyphonic music. As done in Johnson (2017), a natural way to further extend this approach to polyphonic music is to introduce as many *note begin* units as the number of possible pitches in the chosen range.

Overall, the use of note-begin events is quite demanding in terms of input features, which would require doubling the number of input/output features. This could also introduce inconsistencies in the resulting representations; for instance, a note-begin unit for a specific pitch (at the output-prediction layer) can be activated even if the corresponding pitch unit is not.

Modelling and learning musical sequences

The choice of a proper model for music modelling depends on two key aspects: (i) the specific music modelling task to learn, (ii) and the representations used to encode our musical sequences.

To better introduce this part, we first recall the formalisation of the general problem of music modelling. Let us assume that all the notes in our symbolic sequences range from A0 to C8 (88 pitches) and have duration equal to the 8th note. Following metric-based 8th-note quantisation of our sequences, each note is thus contained in a single time slice. Using an absolute local representation of pitch, each musical sequence is encoded as a piano-roll, in the form of an $88 \times T$ matrix $\mathbf{M}$, where T is the number of time steps (varying from sequence to sequence), and 88 corresponds to the chosen pitch range. In this way, a piano-roll is a binary matrix s.t. $m_{p,t} = 1$ if and only if pitch p is active at time step t (the t-th 8th note in the sequence). We can hence define the problem of music modelling on a musical sequence $\mathbf{M}$ as:

$$p(\mathbf{M}) = \prod_t p(\mathbf{m}_t | \mathbf{m}_1, \ldots, \mathbf{m}_{t-1}), \tag{4.1}$$

where $\mathbf{m}_t$ represents the column vector of $\mathbf{M}$ at time step t (i.e. the t-th 8th note in the sequence), $\mathbf{m}_1, \ldots, \mathbf{m}_{t-1}$ is the sub-matrix preceding $\mathbf{m}_t$ and denoting the history of the previous notes. For each musical sequence contained in the dataset, a music model aims at maximising the conditional distribution of the identity of the t-th note $\mathbf{m}_t$ in that sequence, given the identities of all the previous ones $\mathbf{m}_1, \ldots, \mathbf{m}_{t-1}$. This translates into finding the optimal set of parameters θ^* which maximises the probability defined in Equation 4.1 for every musical sequence in the training set $\{\mathbf{M}_1, \ldots, \mathbf{M}_n\}$, each of which has its on length.

This simple formulation allows us to introduce the three main tasks in music modelling: *monophonic* and *polyphonic melody modelling*, and *multi-track music modelling*. Although the formalisation of the prediction problem remains the same, these tasks only differ in terms of the actual configuration of the musical sequences under study. Different configurations introduce or remove assumptions about the data, which in turn contribute to the difficulty of the learning problem. These assumption can be categorised as follows.

- In *monophonic melody modelling*, each column vector in the sequence belongs to a single instrument, and can only have one active pitch per time step. Assuming a piano-roll representation, as in the previous example, this corresponds to considering each column vector $\mathbf{m}_i$ as a one-hot vector for $1 \leq i \leq T$ in a sequence $\mathbf{M}$. In summary, two assumptions are made: (i) single instrument/voice, and (ii) one pitch at a time (regardless of the monophonic nature of the instrument or voice).

- In *polyphonic melody modelling*, the monophonic assumption is dropped, as multiple

pitches can be active at the same time step, but still for the same instrument/voice. This corresponds to considering each column vector in a piano-roll as a multi-hot vector. From a technical viewpoint, this introduces the problem of modelling bi-directional musical dependencies – the joint probability of multiple pitches being activated at the same time step t, conditioned on all past notes. Hence, we are still assuming sequences as melodies (a single instrument score), although they now can be played by a polyphonic instrument.

- In *multi-track music modelling*, no assumption is made about the number and nature of instruments in a music sequence. Until now, we associated each sequence with (the musical part) of a certain instrument. Hence, in the multi-track settings, the encodings should be flexible enough to consider the presence of several instruments (tracks) in a given score. Although the scarcity of multi-track models in the literature makes it difficult to delineate a robust and stable representation of such complex sequences, two simple approaches consist in either combining multiple tracks together in the same matrix, or using a tensor wrapping the piano-roll matrices of each track.

Feed-forward models From a general perspective, all these modelling tasks aim to predict the next token in a sequence – a note (monophonic), a chord (polyphonic), or a group of notes and chords (multi-track), given the context of the previous ones. Nonetheless, not all the *representations of time* introduced in Section 4.1.1, are naturally applicable to any modelling paradigm. The choice of music encodings and model are thus tied together. In particular, special care should be taken when considering a feed-forward model, such as *multi layer perceptrons* (MLP), *convolutional neural networks* (CNN), and their 1-dimensional instances – *time-delay neural networks* (TDNN). In this regard, it is worth taking the following observations into account.

- Using a *global time representation*, where the model is provided with the entire musical sequence and has to chronologically predict each token, requires a masking mechanism. This ensures that the model does not access tokens belonging to future time steps, when predicting a certain element. The task would otherwise be trivial. Furthermore, sequences should also have the same length; common approaches to tackle this issue include: *padding* all sequences with empty or silent tokens, up to the longest duration (that of the longest sequence in a batch); choosing a cutoff duration for all sequences, and performing cropping and padding operations accordingly.

- Choosing a *windowed-time* representation, where the model is fed with a fixed portion of a musical sequence (e.g. a single measure) and is trained to output the next one (e.g. the next measure), has the limitation of limiting the historical context (the temporal conditioning in Equation 4.1, though expressed at the note level) to the previous window only.

As a consequence, a feed-forward model would not maintain any learned representation of the previous musical elements; in other words, this would correspond to implementing a bi-gram model, where each "gram" corresponds to the chosen window.

The greatest limitation of feed-forward models, emerging from our observations, is their inability to process sequences while retaining (and learning to retain) a memory representation (or context) of the previous tokens. Sequences should thus be provided in full – at the cost of additional masking mechanisms, increased memory requirements, and, in some cases, temporal-injection operations; or chunked in frames of the same length, thereby introducing a Markov assumption which is hard to motivate in the context of music. For these reasons, recurrent neural networks are natural candidates for music modelling. In fact, whereas a network using a spatial representation of time (e.g. a TDNN) learns to relate subsequences of predefined length, recurrent networks learn to relate sequence elements at different time scales based on a contextual, stateful representation of the past events.

Recurrent models As detailed in Section 4.2, the pioneering contributions in the literature are based on *Jordan* and *Elman networks* – the earliest recurrent architectures in the field (hence with limited memory capacity). To date, the majority of contributions in music modelling use long-short term memory (LSTM) recurrent networks as the backbone of their models. Considering the same piano-roll example in Equation 4.1, both the hidden state of a vanilla RNN and the cell state of an LSTM can hold the context of the previous elements $\mathbf{m}_1, \ldots, \mathbf{m}_t$, when predicting the next pitch vector $\mathbf{m}_{t+1}$ given the current one $\mathbf{m}_t$ (predicted at the previous step). Having such a flexible way of handling sequences, at inference time, is a desirable property for virtually any sequence model. To name but a few, predicting a sequence of tokens can be done in $O(n)$ complexity (both in time and space), despite the considerable training demands of the backpropagation-through-time algorithm (Section [X]).

The architectural flexibility, the linear time and space complexities of predictions, together with the ability to process sequences of arbitrary length and the native support of music encodings, are the main strengths of recurrent networks. These are all important properties for a music model to address the domain-specific use cases outlined in Section [X]. Therefore, the contributions reviewed in this chapter are all based on recurrent networks. From a systematic investigation of the literature (Section 4.2), most recurrent music models can be categorised into one of the following architectural stereotypes.

Basic recurrent models, using a single recurrent network overall, with one or more RNN cells or LSTM blocks stacked on top of each other, and large hidden vectors in each block-/cell to improve their memory capabilities. Given their simplicity, these models are usually considered as baselines for the evaluation of more complex architecture.

Combined recurrent architectures, using a collection of recurrent networks in a cooperative manner, with each of them modelling a specific specific property, part, or dimension of a musical sequence (e.g. one for chords, another for melody). The output of each sub-network provides a form of conditioning to the others, connected as a chain.

Hybrid architectures, leveraging the strengths of some particular feed-forward architectures, such as CNNs, autoencoders (AE) and restricted Boltzman machines (RBM), to address one or more music modelling issues which will be discussed in Section 4.1.2.

The design of the input and output layers of a recurrent network chosen for a specific music modelling task, are strictly related to the representation used for encoding musical sequences. For instance, when using a piano-roll encoding with the pitch range of a grand piano, at least 89 input/output units would be needed (88 for pitches, and one to denote the end of the sequence). Since we are dealing with a prediction task, which is cast as a classification problem, the output layer is generally a *softmax* or a *sigmoid* layer. The choice of output layer depends on several factors, such as: the specific music modelling task we intend to address, including the assumptions made by the experimenter, the architecture of our network, and the music representations.

As any sequence modelling problem, once a distribution of the next pitch (mono), chord (poly) or group of notes (multi) in music sequences has been learned, as a result of training, a model can then be evaluated. This is usually done by computing the *cross-entropy loss* of the predictions on all the sequences in the test set (the ground truth). Thereafter, if music generation is one of the target use cases, a sampling strategy is considered to sample from the learned distribution(s) – one step at a time (e.g., a single pitch in case of a softmax layer, or possibly more with a sigmoid layer) – with the predictions being fed back to the model. In a recurrent model, this process is repeated indefinitely until a stopping criterion is met (e.g. a predefined number of steps have been generated, or as soon as the `END` token is sampled).

Overview of the main evaluation methods

To the best of our knowledge, one of the major issues in the fields of music modelling and generation is the lack of an evaluation framework for measuring and comparing the performance of music models in both quantitative and qualitative terms. As mentioned before, an example of *quantitative evaluation* of a music model consists in measuring the accuracy of its predictions on a test set of musical sequences. The model is thus evaluated for its prediction capabilities – *how well can a model predict music?* Trivially, having an accurate model will also affect its ability to generate music, although this is out of the scope of such an evaluation. On the other hand, a *qualitative evaluation* of a model capable of generating music (not necessarily a music model) is intended to measure some specific musical properties from a collection of sequences

composed by the model. Performing this type of evaluation allows to address the following question: *how well can a model generate music, as if it was a composer?*

Unfortunately, most of the contributions in the literature seem to neglect the importance of measuring and comparing their models in quantitative terms, in preference to a qualitative evaluation which often requires human involvement. In fact, only a very limited number of contributions compare the predictive performance of their music models with other relevant works in the literature. Whereas a model exclusively targeted for music generation might not need a quantitative evaluation, this consideration does not hold for a music model (a predictive model for music), which is instead expected to serve more tasks other than generation (Section 2.1.2). This tendency seems to be justified by the lack of an evaluation framework, intended as a collection of both quantitative (evaluating the model on the prediction task) and qualitative (evaluating the model on the generated musical sequences) measures to evaluate the overall performance of a music model in a rigorous, affordable and comparable way. If properly designed, this framework could potentially be used as a reference evaluation strategy for future research, enabling and fostering a fair comparison of music models according to standard methods.

The rest of this section outlines the most common quantitative and qualitative evaluation measures and strategies found in the literature. Each of these methodologies is often used in an isolated way, rather than being considered complementary to the others. In addition, there is a tendency in the research community to design novel evaluation strategies to highlight specific properties of their models, with little or no reuse of the existent methods.

Quantitative evaluation measures A quantitative evaluation measure is intended to evaluate the predictions of a music model, and is motivated by the following hypothesis: a model capable to predict music and thus learn the dependencies between the different musical elements, at different levels and granularities, can also capture features related to music theory (e.g. the circle of fifths). The extent to which a model can learn musical features is thus assumed to be proportional to the quality of its predictions. The first three measures reported below are common in general prediction problems, whereas the last one is domain specific.

Log-likelihood (or negative log-likelihood) of the model's predictions on the test set. In the few cases a quantitative evaluation is provided, this is often the preferred way of measuring the predictive capabilities of a model on unseen sequences.

Frame-level accuracy was proposed by Boulanger-Lewandowski et al. (2012) as an alternative way, brought from speech recognition, to measure the accuracy of predictions at a frame level, expressed in absolute (e.g. 10ms) or relative (e.g. a measure) windows.

F-measure, *precision*, *recall* and *perplexity* are common evaluation measures for classification tasks, which were also proposed for music modelling by Ycart et al. (2017).

Measures of tonal tension. As musical tension is a central concept in music theory and perception, using specific measures of musical tension, such as *cloud diameter* (a measure of dissonance), *cloud momentum* (a measure indicating changes in tonality), *tensile strain* (a measure indicating tonal distance from the global key of the piece), has been demonstrated to be an effective indicator of stable tonal predictions (Chuan & Herremans, 2018).

As argued by Chuan and Herremans (2018), general machine learning measures are not sufficient indicators of the musical features learned by the model unless other music-based measures, such as the tonal tension, are used to complement this evaluation. Furthermore, these measures depend on the granularity of the chosen representations (e.g. the quantisation level), meaning that comparisons between music models are only possible if one of these conditions holds: (i) models use encodings sharing the same sequence length; or (ii) only the average score per sequence is of interest (which may be further aggregated to yield a single indicator for the whole test set). Otherwise, more granular comparison of these measures (e.g. at the token level), would not directly be possible. In fact, if two models use encodings that are not directly comparable, their sequence elements (tokens) would not hold the same musical content (regardless of the way content is encoded).

Qualitative evaluation strategies It is important to remember that music modelling and music generation are two different problems governed by different requirements. Whereas music modelling aims at capturing musical patterns and knowledge from musical sequences by learning to predict music, methods for automatic music generation exclusively aim at sampling novel compositions, regardless of how the model is trained. Nevertheless, a music model is also a model for generating music, although the same cannot always be said for a generation model (i.e. music modelling implies music generation, but not viceversa).

This observation enables the evaluation of music models using the same strategies for music generation, applied directly on a collection of musical sequences generated from the model. The most common measures and strategies for qualitative evaluation are reported below.

Statistical comparison. It consists in comparing the descriptive statistics of the generated musical sequences with those computed on the training data. This gives a weak measure of the extent to which generated sequences resemble those contained in the training set – a plagiarism score in a way (B. L. Sturm et al., 2016). A high level of similarity with the training data might also be interpreted as an indicator of overfitting, or suggests that the sampling method needs to be fine-tuned.

Composition evaluation. The purpose of this evaluation consists in assessing the quality and the plausibility of the generated sequences in terms of music rules and theory, as a composition teacher would do with the work of a student. This can be done via computational measures derived from music theory, or involving a community of music experts. However, given the scarcity of computational measures, such a manual evaluation of generated compositions is a time-consuming task requiring a high level of musical expertise. Indeed, even a professional musician might not feel confident to give detailed composition feedback.

Human Evaluation. This group collects the two most common evaluation strategies used in the field of music generation, which are explained as follows.

Turing test (aka *discrimination test*). A group of listeners with different musical background is presented with musical pieces either composed by humans or generated by a model. Listeners are asked to discriminate among these groups, which basically corresponds to answering the question: *was this piece composed by a human or by a machine?*. Whereas a model generating music that cannot be clearly distinguished from human work is a positive indicator of its generative capabilities, this "pass-or-fail" methodology does not allow comparisons with other works in the literature.

Blind comparison of musical sequences generated by different models. These sequences (usually no more than three per model under analysis) are assessed by a group of human listeners with respect to different criteria. The goal is to measure to what extent each piece shows certain properties that would be expected from a real composition. This approach thus allows to rank each model according to the obtained measures. From a critical perspective, this methodology is sensitive to potential biases emerging from the selection of tracks to include the experiment.

Most of the works using human evaluations rely on paid services, such as *crowd-source platforms* (e.g. Amazon Mechanical Turk), where participants receive a fee for their evaluation, and may not be easily filtered according to a desired degree of specificity; or on *web-based platforms* anyone can access to provide feedback. Hence, these strategies may not be accessible to all researchers, and do not always guarantee an adequate level of control needed for the experiments.

To conclude, it is worth remarking that these evaluation strategies are currently used in an isolated manner – only one or a few of them is chosen, which often does not include a comparison with other works in the literature. As advocated by A. Huang and Wu (2016), more work should be done in developing better evaluation metrics: *only then will we be able to train models that are truly able to compose original music*. The design of a comprehensive

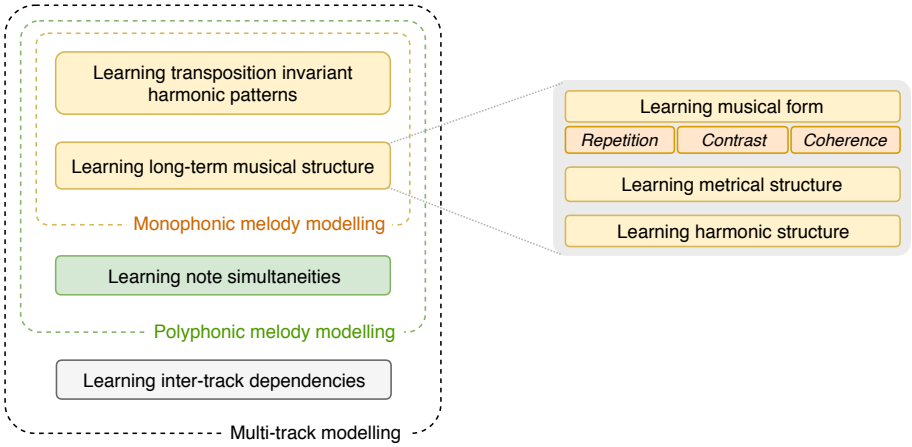


Figure 4.1: Task-based overview of the major technical challenges in music modelling.

evaluation framework, providing measures and strategies both at the quantitative and qualitative levels, would also foster the introduction and the comparison of novel music models.

4.1.2 Technical challenges: a task-based overview

Now that the main properties of the problem have been presented, we can proceed with its major technical challenges. The degree of difficulty behind each of them, inherently related to the complexity of the music signal, also depends on the style and genre of the specific collection under study. To present these issues, we organise them in a hierarchy, which emphasises how the three modelling tasks progressively add complexity to the prediction problem.

As can be seen from Figure 4.1, the two major technical issues in monophonic music modelling – learning *transposition invariant features* and *long-term musical structure*, are shared with all the other modelling tasks. As better clarified later on, learning musical structure at different temporal spans is currently considered as the main open problem in the field. At the next level in the hierarchy, the task of polyphonic melody modelling introduces the issue of learning *note-simultaneities* (chords), while retaining past musical context. Finally, if musical sequences are not limited to a single instrument or voice, multi-track music modelling adds the challenge of learning inter-track dependencies. This last task is thus the hardest to address, as it includes all the open challenges in music modelling.

Learning long-term musical structure As discussed in the introduction (Section 1), music is not merely a sequence of notes or sounds, but it entails a rich hierarchical structure of its elements at different levels and musical dimensions. For a model, this necessitates to relate musical segments at different time scales, hence the ability to detect and retain them across the whole piece. In sequence modelling words, this requires considerable memory capabilities,

depending on the structural complexity of the music collection under study.

As observed by Eck and Schmidhuber (2002b), the challenge of learning long-term musical structure is generally attributable to the well-known vanishing gradient problem of recurrent networks (Bengio et al., 1994), which in turn translates into their inability to capture long-term dependencies in sequences. In the context of music, long-term dependencies are at the heart of what defines a particular style, with events spanning several notes (or bars) contributing to the formation of metrical and phrasal structure (Cooper & Meyer, 1963). A simple example of this is a chord progression in rock-and-roll music, where the same chord is repeated for several bars.

Evaluating the extent to which a music model has learned musical structure is usually done by manually analysing the compositions generated by the model after training. This consists in inspecting the possible structural properties emerging from the synthetic compositions. More precisely, a checklist for the experimenter may include the following structural goals.

Learning global structure (music form). Musical ideas are presented and organised at different levels of granularity, with the corresponding segments being coherently *repeated* and *contrasted* across the piece at different time scales. For instance, a succession of notes can be grouped in a *figure* or *motif* – the shortest musical ideas; in turn, a group of figures/motives can make up a *phrase* – a more articulated musical idea with proper musical identity; again, different phrases can be combined to form *melodies*, *periods*, *sections* and so forth. Therefore, these structural components, presented in ordinal fashion (figures, motives, themes, phrases, periods, and sections), show hierarchical dependencies with each other. Their organisation in a piece is characterised by the following aspects.

Repetition, a fundamental aspect of music, which has even been found in music sang or played by children (Eck & Lapalme, 2008). In a composition, the first occurrence of each segment generally presents a musical idea, which is then conventionally imprinted by repetition. For example, motives are generally repeated within a phrase. At a coarser level, sections may be repeated in a musical piece (e.g. the AABA form).

Repetition poses a challenge for artificial neural networks and graphical models, such as Hidden Markov models (HMMs), as it requires explicit memorisation. In particular, Eck and Lapalme (2008) argue that a vanilla RNN could learn to repeat a fixed number of learned patterns, but it could not learn to repeat an arbitrary sequence, because it has no way to implement content-addressable memory. Similarly, an HMM would require an explosive number of states to repeat an arbitrary pattern.

Contrast, another important ingredient used by composers to bring novelty in a piece, and ensure that the composition is not perceived as too repetitive and monotonous to

listeners. This is usually achieved by alternating different musical ideas at different time scales, thereby promoting variation and potentially increasing musical tension.

Coherence, considered as the balance between repetition and contrast, and needed to ensure that music does not diverges, say from the main chord progression or melody. Repetition creates a sense of familiarity, contrast brings novelty and may increase musical tension, which eventually can be resolved by re-proposing a familiar idea. This interplay is virtually the expression of the composer's intentions and her own sense of coherence for a certain composition. Although coherence can be established in different ways, a prevalent approach is to rely on harmonic structure. In this context, a music model that effectively learned to detect the harmonic structure of a piece, has the potential to generate musically coherent compositions.

As observed by A. Huang and Wu (2016), most of the current music models suffer from the problem of learning global structure, and their generations are poor in that regard. More precisely, the resulting musical samples are too repetitive from a local perspective (e.g. the same short musical idea is repeated indefinitely with little variation), or too contrasting at different levels, thus indicating the absence of long-term repetition and coherence. Overall, excessive repetition of short musical ideas and variation are the most common scenarios.

Learning metrical structure. The notion of *meter* in music can be described as the sense of strong and weak beats that arises from the interaction among hierarchical levels of sequences having nested periodic components (Eck & Lapalme, 2008). This hierarchy is implied in Western music notation, where different levels are indicated by kinds of notes (whole notes, half notes, quarter notes, etc.) and where bars establish measures of an equal number of beats (e.g. most contemporary pop songs are built on four-beat meters, where the first and third beats are usually emphasised). Simply put, the direct connection with music notation makes meter the backbone of a composition. Therefore, both global and harmonic structure largely depend on meter. In the former case, this can be easily seen from the tendency of music form to be segmented at the metrical level. For this reason, if the meter of a piece is provided to a music model, this would facilitate the detection of structural boundaries, such as the location of chord changes or section endings.

Learning harmonic structure. Given that harmony is often referred to as the *"vertical dimension"* of music, we should not confuse this term with the problem of learning note-simultaneities in polyphonic melody modelling. In the context of structure, instead, we focus on the harmonic dependencies between successive pitches. In particular, given a musical sequence, the prediction of the next pitches should conform with the previous notes in

relation to the tonality of the piece (i.e. key and scale) at a certain moment (the tonality can change during the piece), which impose the observance of certain intervallic conventions.

Whereas learning global structure (or form) is currently one of the hardest challenges in music modelling and generation, capturing metrical and harmonic structure also involves dealing with hierarchical dependencies. Considering the interconnection between all these musical dimensions, a predictive model can be facilitated to learn the former if it is provided with at least one of the latter (as a form of conditioning signal alongside the input sequence).

Learning transposition-invariant features. Predictive models for music are trained to find relationships between past and future occurrences of pitches, which, most of the times, are encoded in terms of their absolute values. As outlined in Section 4.1.1, this design choice poses the problem of learning to relate musical events regardless of the tonality of pieces – a property commonly referred to as *transposition invariance*. In other words, most music models are not transposition-invariant (Johnson, 2017). In these models, each note is represented as a distinct element in a vector, so there is no way for the network to generalise intervals and chords: for instance, any musical relationship between C and G, must be learned independently from the relationship between C# and G#, as well as any possible transposition of them.

In contrast, several music theoretical constructs that can formally describe such relationships are expressed in relative terms, such as diatonic scale steps, and cadences to name but a few (Lattner, Grachten, & Widmer, 2018). When music is written in a particular key, notes are interpreted in relation to that key, and chords are often classified based on their position in that key (e.g. using Roman numeral position). Transposition – shifting all notes and chords into a different key, changes the "absolute position" of notes but not their musical relationships.

The discrepancy between the relative nature of music regularities and the use of absolute pitch representations adds difficulty to the music modelling problem. In practice, this can lead to high sparsity in the encoded data, increased model sizes, and reduced generalisation capabilities (Lattner et al., 2018). Common approaches to address the transposition invariance problem include leveraging interval encodings and data preprocessing techniques.

Using relative representations of pitches. This way, a model does not need to learn the same musical regularities for every possible key, but only features related to the scale (e.g. major, minor, blues). However, as observed in Section 4.1.1, interval representations also come with a few issues, such as returning to the original tonality in case of a mistakenly predicted note (Todd, 1989). A more sophisticated approach, is to design the architecture of a model or defining a training strategy to learn transposition invariant features regardless of the encodings (Johnson, 2017; Lattner et al., 2018). In either case, we would be still providing prior musical knowledge to the model (either in the encodings or in the architecture).

Transposing musical sequences to the same key. If every piece in the dataset includes references to its key, a simple approach is to transpose all tracks to the same (predefined) key. Although this allows using absolute representations, in most cases it is very unlikely to assume that key information is explicitly available. In addition, this also reduces the generalisation capabilities of a model, as it will be restricted to learn musical regularities related to a single key. In other words, the model will not generalise to music in all the other keys.

Transposing musical sequences to all keys. Presenting a model with all the possible transpositions of every piece in the training set – a *data augmentation* strategy per se – can be considered as an implicit way to let the model learn transposition-invariant features. In fact, data augmentation provides a simple way of encoding domain-specific knowledge in virtually any machine learning algorithm. Not only this creates a more diversified training set, which is generally expected to improve generalisation, but also, altering the existent data to emphasise some invariant features (which are still preserved after the alteration process), is an implicit way to encourage the model to grasp such regularities.

In summary, using interval representations is the easiest way to learn transposition-invariant features, but comes at the cost of unstable tonal predictions in case of mistaken pitches. Conversely, explicitly learning interval representation, as part of the training process, might require considerable interventions at the architectural level (Johnson, 2017), or extensive fine-tuning and pre-training of specific parts of the model (Lattner et al., 2018). On the other hand, transposing all pieces to the same key circumvents the problem but limits the generalisation capabilities of a model. Finally, transposing every piece to all keys is a simple yet effective solution, which might however be computationally demanding for large datasets.

Learning note simultaneities The additional challenge introduced by polyphonic melody modelling is the *prediction of chords*, or more generally, music composed for a polyphonic instrument (e.g. the piano). In this context, multiple pitches can be played at the same time step, and their occurrence is not merely casual, but generally adheres to certain harmonic regularities – the *vertical dimension* of music – promoting the consonance of the resulting chords.

In technical terms, since a polyphonic melody can be considered as a sequence of high-dimensional objects, the conditional distribution is very often multi-modal . Indeed, the occurrence of a particular note at a particular time modifies considerably the probability with which other notes may occur at the same time (Boulanger-Lewandowski et al., 2012). Notes thus appear together in correlated patterns, also known as *simultaneities*.

To formally describe the problem, let us consider a piano-roll matrix $\mathbf{M}$ of a particular musical sequence (the same notation introduced in Section 4.1.1). Polifonic melody modelling generalises Equation 2, as it requires approximating the following conditioned joint probability:

$$p(\mathbf{M}) = \prod_t p(m_{p=1,t}, \dots m_{p=88,t} \mid \mathbf{m}_1, \dots, \mathbf{m}_{t-1}), \tag{4.2}$$

where $m_{p=1,t}$ denotes whether the first pitch in the range (A0) is active, and $\mathbf{m}_1, \dots, \mathbf{m}_{t-1}$ is the temporal conditioning (as in the monophonic case). Although several harmonically inadmissible simultaneities could be ruled out, the formulation still accounts for the possibility of having an arbitrary number of pitches active simultaneously. The resulting musical relationships can thus be considered as *bidirectional music dependencies*, spanning across both the *temporal dimension* (horizontal) as well as the *harmonic dimension* (vertical) of music. Common approaches aiming to reduce the complexity of the polyphonic settings are outlined as follows.

Extending the vocabulary, to account for all the possible combinations of a predefined number of pitches. For instance, given the 88 pitches of a gran piano, and assuming to support chords made of up to 5 simultaneous pitches, a feature vector would contain $89 \times 88 \times \dots \times 85$ elements in total[1]. To limit the size of the vocabulary, it is also common to discard all those combinations of pitches which are generally unlikely to occur as chords. This approach is usually motivated by the statistics of the training data (chord distribution), and allows the experimenter to embed prior knowledge of music theory by ruling out chords with unrealistic intervals (e.g. C-C#-D-D#-E).

Unrolling polyphonic events in time, to represent chords as sub-sequences where the time step does not change. This is simply done by introducing special events, usually denoted as *start-step* and *end-step* events, to delimit the constituent pitches of a chord sub-sequence.

Modelling the conditional distributions of different random variables, each representing one element of polyphony according to a predefined upper bound of simultaneous notes. For instance, if only 5 pitches can be played at the same time, 5 different random variables would then be introduced on the same vocabulary. The time-step probability of each random variable would be conditioned on the previous musical context – the sub-sequences of the previous elements drawn by all random variables until the current time step.

With the first two approaches, the polyphonic modelling task is simply cast to the monophonic case, as note simultaneities would be encoded as individual events. From a critical perspective, extending the input vocabulary by combining multiple pitches in a single event would not allow the model to relate the constituent pitches of a chord (e.g. a C2 major chord would not encode any relationship with the vector elements corresponding to the C2, E2, G2

[1] For simplicity, the *no-note* event is also included in this calculation, which makes it possible to represent chords with less than 5 pitches, as well as single notes. In general, the chosen pitch representations will determine the actual number of elements – the vocabulary of musical events – that will be combined in a feature vector.

pitches). In addition, the large vocabulary required by the former approach would also produce sparser encodings. Unrolling polyphonic events, instead, can dramatically extend the encoded musical sequences, thus exacerbating the problem of learning long-term dependencies.

Learning inter-track dependencies Finally, as the assumption of modelling single-track music pieces is discarded, multi-track music modelling is clearly the most general task in music modelling. As can be seen from Figure 4.1, this task includes both the monophonic and polyphonic problems, and introduces the open technical issue of modelling the dependencies among the different instruments or voices in a multi-track musical sequence. Therefore, a multi-track model should be able to address all the other issues presented so far, while learning to relate musical content from different instruments. In other words, an inter-track model should predict music as a director would do for an orchestral performance.

4.2 Literature review of music modelling methods

4.2.1 Monophonic melody modelling

Also referred to as *melody modelling*, or *monophonic music modelling*, this task focuses on single-track pieces where the main voice is monophonic. Hence, a monophonic model is trained to predict the next pitch or note in a sequence, given the temporal context of the previous tokens.

As detailed in Section 4.1.2, the main technical issues for this task are: learning transposition invariant features and learning long-term musical structure. The literature review proposed in this section categorises monophonic models according to the taxonomy reported in Figure 4.2. The taxonomy contextually relates the technical challenges (the monophonic portion of Figure 4.1) with the kinds of methods specifically intended to address them.

As can be noticed, most methods in the literature of monophonic models focus on the challenge of learning long-term musical structure. To achieve transposition invariance, instead, monophonic models tend to use one of the general approaches described in Section 4.1.2. In short, prior knowledge is implicitly provided to the model through data manipulation – by transposing all pieces either to the same key, or to all the possible keys within the chosen pitch range.

Prior knowledge is indeed a central aspect in music modelling, and the methodologies reported in Figure 4.2 can also be grouped according to whether the resulting models or the adopted representations are intentionally designed to include implicit musical concepts. In particular, we consider the following categorisation.

Learning musical features without prior knowledge. Models belonging to this category are challenged to learn musical features without any kind of prior knowledge encoded in the

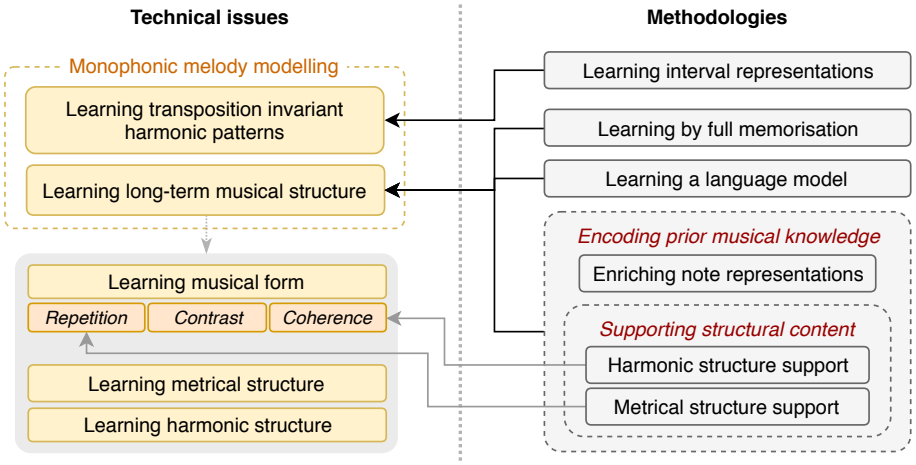


Figure 4.2: Technical issues and current methods in monophonic melody modelling.

model or in the representations. If the goal is to achieve transposition invariance, the tendency is to use composite architectures that can learn interval representations and how to use them. Conversely, methodologies for learning long-term musical structure include using textual representations of music sequences – so that traditional language modelling techniques can be exploited, or learning to fully memorise a short collection of musical pieces.

Encoding prior musical knowledge. The contributions within this group are based on providing prior musical knowledge in the form of enriched note representations, or by conditioning the model on specific structural content of the musical sequences. More precisely, the former methodology is based on the intuition of encoding harmonic concepts in pitch representations, and music perception concepts within duration representations. Conversely, contributions based on the second methodology do not extend the input representations, but they condition the prediction of the next music event also on structural content, such as harmonic (e.g. chords) or metrical (e.g. bars) information.

The remainder of this subsection reviews the most relevant contributions in each category at both conceptual and technical level. Each work is presented is such a way as to highlight its main properties – *representations*, *model* and *learning strategy*, together with the proposed *evaluation*. This is also complemented with a concise but technically detailed critical overview of the major limitations of each work, so as to provide motivations for further research.

Encoding prior musical knowledge by enriching note representations

As outlined before, the contributions following this methodology are based on the hypothesis that providing a model with musically enriched representations can improve its ability to learn

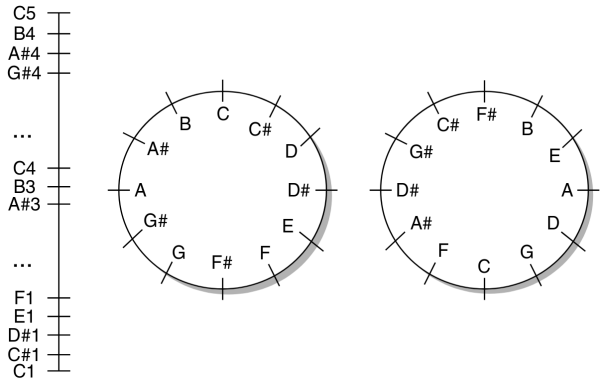


Figure 4.3: Overview of PHCCCF representations: pitch height, chromatic circle, circle of fifths. Taken from (Mozer, 1994).

musical properties and regularities, such as harmonic and metrical structure. The main contributions implementing this idea are the works by Mozer (1994) and Franklin (2006), where the latter can be considered as an extension of the former.

The pioneering work of Mozer (1994) introduced CONCERT – a recurrent network which is said to learn coarse musical structure at the phrase level (Franklin, 2006). From an architectural perspective, the music model consists in a vanilla recurrent neural network processing musical sequences at the note level. The peculiarity of CONCERT lies in its sophisticated musicology-inspired approach to represent note pitch and note duration, which are treated separately.

Representation of pitches. Each pitch is encoded based on its fundamental frequency, chromatic class, and position in the circle of fifths. Known as PHCCF, this psychologically based representation of musical notes derives from (Shepard & Levitin, 2002). As illustrated in Figure 4.3, the chromatic circle (CC) and the circle of fifths (CF) are used with a linear octave value called pitch height (PH). Six digits represent the angular position of a pitch on the CC, and six more encode its angular position on the CF. PH is represented as a single scalar input that maps the 48 pitch values between C1 and C5 to values between 1 and 20.

Representation of note duration. As musical sequences are processed note by note, the duration of a note is represented as a separate feature. As illustrated in Figure 4.4, the duration representation is analogous to that of pitches, as durations are represented as positions on three scales, with a quarter note being divided into 12 subdivisions. The angular positions on each of the *mod 4/12 circle* and the *mod 3/12 circles* is determined by the remainder after dividing by 12, then by 4 or 3. The duration height is the amount of duration divided by 12.

In the same vein of Mozer (1994), and inspired by the work of Eck and Lapalme (2008) for learning longer structural regularities in music, Franklin (2006) introduced a monophonic

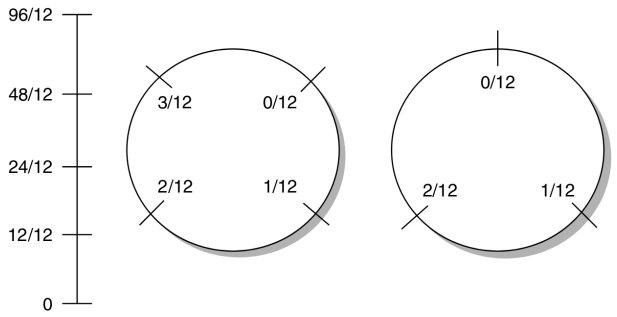


Figure 4.4: Duration Representation: Duration Height, Mod 4/12 Circle, and Mod 3/12 Circle. Taken from (Mozer, 1994).

model for jazz melodies and accompaniments. The author demonstrates that an LSTM fed with musical sequences encoding domain-specific music knowledge, is facilitated to learn longer jazz pieces. His work can be considered as an extension of CONCERT with respect to both model architecture and note representations.

Pitch representations. Franklin's representation of pitch, named *circles-of-thirds representation*, builds upon Mozer's localised binary and CCCF representations originally proposed for chords. Figure 4.5 shows the four circles of major thirds (a major third being four half steps between pitches) and the three circles of minor thirds (a minor third being three half steps). The top row includes the set of circles of major thirds, each reading counter-clockwise: E is a major third above C, G# is a major third above E, and so forth. Similarly, in the second row, E-flat (assumed to be equivalent to D#) is a minor third above C, F# is a minor third above D#, etc.

The resulting pitch representation needs seven bit as follows: the first four bits indicate the circles of major third in which the pitch lies; analogously, the last three bits are associated to the circles of minor third. Unlike PHCCCF, where the encoding of a pitch corresponds to its angular position in the circle, this representation uses the index number of the circle. For instance, C is encoded as 1000100 (as it appears in both the 1st and 5th circle), indicating major circle 1 and minor circle 1. Hence, unlike PHCCCF, pitches that are half a step apart do not have similar representations – a property which is instead desirable from the harmonic perspective, as a half step error often sounds out of place. Nonetheless, a pitch with one bit out of place would still share a common major or minor interval with the one intended. To conclude, the octave information is encoded as a separate feature.

Duration representation The representation of note duration extends Mozer's work to encode notes with complex length, which are needed for modelling musical performances. The author intends to move beyond score-based durations, and let the model learn human-like variations in duration which are common in jazz music. More precisely, a quarter note is split into 96

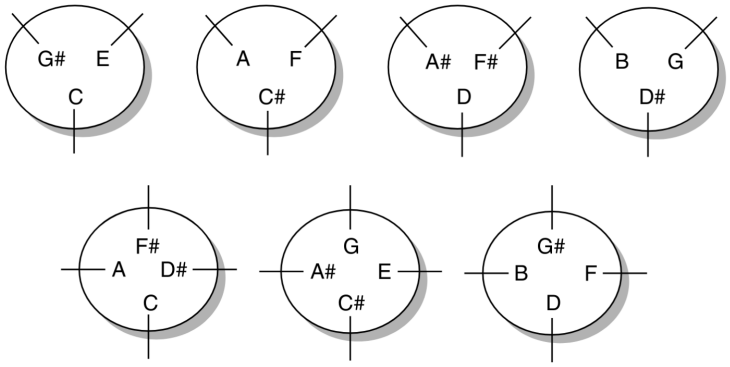


Figure 4.5: Circles-of-Thirds Pitch Representation: circles of major (top) and minor thirds (bottom). A pitch is uniquely represented via these circles, assuming octave and enharmonic equivalence. Taken from (Franklin, 2006).

subdivisions, a standard called "ticks" in the Musical Instrument Digital Interface (MIDI) standard digital protocol. In this way, a whole note, dotted half, half, quarter, dotted eighth, eighth, eighth triplet, sixteenth, sixteenth triplet, thirty-second, thirty-second triplet, and sixty-fourth are represented as 384, 288, 192, 96, 64, 48, 32, 24, 16, 12, 8, and 6 clicks, respectively.

To implement a binary encoding, while minimising its dimensionality, a modulo-based representation of 16 bits is adopted. In particular, the 16th bit is activated if the duration of a note divided by 384 is greater than or equal to 1. The 15th bit is 1 if the remainder of the former division, further divided by 288, is greater or equal to 1, etc. For instance, a dotted quarter note is expressed as $96+48$. Not only this representation allows to represent traditional score-notated durations, but also human-performed approximations or improvised durations.

Evaluation. The author qualitatively evaluated the LSTM network trained on the proposed representations on different musical tasks. One such task, called AABA melody, tested the model's ability to reproduce a specific 32-note melody in AABA form, given the first notes as input. This memorisation task was accurately performed by the LSTM, even though the experimental setup presents several assumptions: note durations are ignored, and the sequence is relatively short to conclude any long-term capabilities. Besides the structural test, the evaluation performed on a single but longer musical sequence demonstrated the advantage of using circles-of-third representations with modulo-based durations for modelling monophonic music.

Limitations. Although the authors demonstrated that their models can capture musical structure, the actual patterns and regularities learned are mostly local rather than global. In other words, what they seem to have achieved with these models can be interpreted as learning short-term musical structure. Indeed, Mozer argues that CONCERT succeeds in modelling local properties of a melody (e.g. step-wise motions), but still fails to capture longer structures –

"*while the local contours made sense, the pieces were not musically coherent, lacking thematic structure and having minimal phrase structure and rhythmic organisation*".

Encoding prior musical knowledge by supporting structural content

The general methodology shared by the contributions in this category is based on providing a model with specific structural properties, in order to facilitate the process of learning global structure and enforce repetition and coherence (c.f. Section 4.1.2). More precisely, as illustrated in Figure 4.2, two specific approaches can be identified: (i) conditioning the model with harmonic content in order to promote musical coherence and tonal stability; (ii) conditioning the model with metrical content related to previous bars to highlight repetition boundaries.

Supporting harmonic structural content. By leveraging supplementary harmonic information in lead sheet music, the main goal of these methods is to learn music form, and promote the tonal coherence of the model's predictions. Here, this is achieved by using a recurrent network with considerable memory capacity, while explicitly conditioning the prediction of the next musical event also on the previous chords in the sequence (provided as additional input). Supporting chords for the prediction of the (monophonic) melody thus emphasises the harmonic structure of the piece, which in turn promotes the tonal coherence of the predicted notes.

A central contribution implementing this idea is that of Eck and Schmidhuber (2002b), who complement a monophonic model with a separate model specialised in the prediction of chords. Basically, a recurrent network is first trained to predict chords. The predictions of the chord model are then provided to the monophonic model, as conditioning signal, to facilitate the prediction of the next melody events in the same musical sequence.

Music representation. In contrast to the sophisticated encodings outlined before, Eck and Schmidhuber use simple piano-rolls encodings to represent a chromatic octave. The pitch range of the input musical sequences is set to 13 pitches for the monophonic model, whereas only 12 chords are considered for the chord model. The representation of time is thus straightforward, with a single vector representing a slice of time corresponding to the 8th-note. As advocated in (Gers & Schmidhuber, 2000), this encoding method is preferable for LSTMs because it forces the model to learn the relative duration of notes, making it easier for the counting and timing mechanisms to work. To train and evaluate their method, a collection of 12-bar blues is chosen. Given the quantisation to the 8th note and assuming a 4/4 time signature, each piece would have 8 notes per bar, hence 96 time steps in total.

Model. As local (one-hot) encoding are used, the input layer includes 13 units for pitches in the chromatic octave, and 12 units for chords. The hidden layer consists of two blocks of 8 LSTM cells each: one block specialises in modelling the melody and the other focuses on harmony. The authors also include recurrent connections from the *melody block* to the *harmony block*, but not viceversa. This design choice indeed reflects the actual nature of the proposed methodology – conditioning the prediction of the melody line on chords, as a simple strategy to provide harmonic structure to the monophonic model.

Evaluation. To evaluate the model, two different experiments were performed: *learning chords*, needed to verify whether chord structure can be induced in absence of melody; and *learning melody from chords*, based on the assumption that elements of harmonic structure lead the melodic line – a property that is particularly emphasised in blues/jazz music. This also allowed the authors to generate melodies that never diverge far from the original chord progression, thus demonstrating improved coherence and clear harmonic structure in the compositions.

Limitations. From a critical perspective, the model operates on a limited set of pitches and chords, which are quantised at a very coarse level (the 8th note). In addition, the model architecture and the proposed training strategy assume that chords are always available, as the conditioning system fully relies on such information. However, though chords are generally found in lead sheets – a common music format in blues/jazz music – most symbolic music collection do not provide harmonic information. Hence, the use of simple representations and the mandatory chord conditioning undermine the application of the music model to other domains.

Supporting metrical structural content The methodology characterising the work of Eck and Lapalme (2008) is also based on encoding prior music knowledge by providing structural content to the model. Here, instead of providing harmonic content, the model is conditioned on the past notes found at metrically-spaced time steps, e.g. collectively at the previous 4, 8, and 12 measures. As anticipated in Section 4.1.2, knowing the meter of a piece in advance, can facilitate the prediction of structurally relevant events, such as the location of chord changes and repetition boundaries – as music tends to repeat in proximity of metrical boundaries.

Based on these motivations, Eck and Lapalme implemented the metric-based conditioning in an LSTM network by using time-delay connections. There are no particular differences with the musical representations adopted in the previous work; musical sequences are still quantised to the 8th note and local encodings for chords and pitches are used as before. This time, however, the model is trained on Irish folk music transcriptions, which are all transposed to the same key in order to circumvent the transposition invariance problem.

Model. The authors explicitly provide metrical information to an LSTM network in the form of delayed input connections (from previous sequence elements). Given the quantisation of 8 time steps per measure, and assuming that all pieces have time signature of 4/4, the models receives time-delayed copies of the input at $t-15$, $t-32$, $t-63$ (corresponding to 2, 4, and 8 measures respectively) to predict $(t+1)$-th musical event. This way, it is easier for the LSTM to correlate metrically-relevant musical patterns, as these lags are given special salience Nonetheless, the network can still attend to other lags via its recurrent connections.

Though an LSTM can in principle identify these lags without any conditioning, this is very difficult to achieve in practice. As described by Eck and Lapalme, an LSTM searches for repetition at any lag – similarly to identifying strings in the A^nB^n context free grammar where n is unbounded. By providing metrical information to the network, in the form of delayed inputs, this unbounded search (for all the possible n) is cast to a bounded search for A^kB^k strings, where k now corresponds to one of the predefined metrical lags. The LSTM still looks for any repeating structures, but the lags at which it will likely look are intentionally biased towards the metrical boundaries. At the same time, the LSTM can still search at other lags in the input using its own dynamical gating mechanism (the same mechanism used for the A^nB^n problem) to identify any relevant long-timescale dependencies that do not align with metrical boundaries.

With the provision of metrical information, the model demonstrated better abilities to capture some of the repetitive structure characterising folk music. The resulting model is said to responds to musical structure on three levels: *local structure*, learned using standard feed-forward connections; *hierarchical metrical structure*, acquired via the time delay connections; and *non-hierarchical long-timescale structure*, which is learned using the LSTM. However, the choice of measures to look back, needed to implement time delayed connections, can only be made if all pieces share the same time signature. In practise, this assumption does not always hold, as metrical structure can change from piece to piece, even within the same musical style.

Learning and processing interval representations

Differently from the previous methodologies, this approach specifically addresses the problem of capturing transposition-invariant features (Figure 4.2) without encoding any prior musical knowledge in the model architecture, nor in the music representations. This methodology generally provides an elegant solution for learning transposition-invariant musical regularities, without transposing the sequences either to the same key or to all the possible ones.

In the context of monophonic music modelling, the most relevant work in this category is that of Lattner et al. (2018), proposing a combination of different neural network architectures to learn *interval representations*. Their contribution comes from the observation that RNNs

using symbolic representations with absolute pitches usually fail to generalise over musical concepts commonly perceived in terms of relative distances between pitches (e.g., melodies, scale types, modes, cadences, or chord types). As observed by the authors, this also represents an obstacle for an RNN when it comes to detect repetition in the context of musical form. For instance, the self-similarity of a piece should also account for variations of sections rather than full repetitions only, as repeated sections can often occur as transposed versions of a section.

Model. Using a quantisation to the 8th note, Lattner et al.'s model can learn both interval representations from absolute pitch sequences (simple piano-rolls encodings) and temporal dependencies between their intervals. In particular, a *gated autoencoder* (GAE) is first trained to learn a latent representation of intervals. Then, an RNN directly operates on these interval representations in order to learn temporal dependencies. Learning musical intervals thus allows the architecture to operate explicitly on musical textures and reduces the sparsity of absolute representations. Moreover, as the representations learned in the first step are transposition-invariant, the resulting recurrent GAE (RGAE) requires fewer temporal connections than a traditional RNN. Temporally processing intervals is also said to alleviate the problem of detecting repetitions, as the model would interpret that as a copy-and-shift task. This is performed by repeatedly applying a constant interval to events occurring at constant time lags in the past.

Evaluation. The proposed approach is evaluated on the music modelling problem, as described in Section 4.1.1, using the Essen Folk Song Collection (EFSC) (Schaffrath & Huron, 1995). More precisely, the authors designed two experiments to demonstrate the benefits of the RGAE architecture: the first one indicated increased accuracy for the task of music prediction, following a comparison with a simple abolute-pitch model; the second one, utilising a manipulated version of the dataset, clearly demonstrated that RGAE can outperform a simple recurrent network if stressed on the issue of musical repetition (copy-and-shift task).

Limitations. In the EFSC dataset, melodies are represented as series of pitches and note durations are ignored. Time is thus homogeneous from note to note, and this is certainly a factor which facilitates learning interval representations. However, disregarding note duration is both unrealistic and uncommon in processing music, regardless of its pitch representation.

Learning musical properties by full memorisation

The methodology proposed by Todd (1989) consists in using a Jordan network – one the precursors of modern recurrent neural networks, to implement a model which is intentionally trained to memorise and reproduce a limited set of musical pieces. The author argues that a model

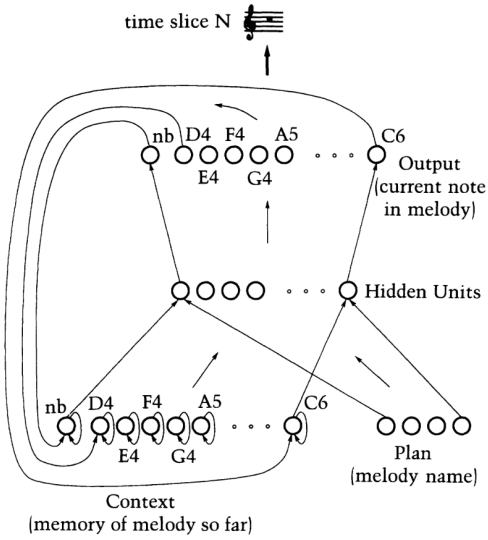


Figure 4.6: Illustration of the Jordan recurrent network introduced by Todd.

capable to fully reproduce a collection of pieces is indeed a has potentially learned structural properties of the music signal and can be used to generate new compositions.

Music representation. The Jordan recurrent network receives a relatively small collection of musical sequences encoded as piano-rolls with quantisation to the 8th note. To represent consecutively repeated notes with piano-roll encodings, a "*note begin*" event (denoted as *nb* in Figure 4.6) is used to denote the onset of a note. Besides the note begin, 14 events are used to represent pitches from D4 to C6. Differently from most representations for music modelling, the input encodings are not the same as the output ones. In fact, the input encodings also include 4 events, called *plan inputs*, to identify each of the 4 34-notes melodies to memorise.

Model. As illustrated in Figure 4.6, the output of the network is fed back to the input layer, with a recurrent connection on each input unit. The actual input is a decaying average of the most recent output values, thereby providing a decaying memory of the melody. This is indeed a peculiarity of Jordan networks, which use is motivated by the fact that they tend to be easier to interpret. In contrast, as explained in Section [X], vanilla recurrent neural networks use feedback connections from the hidden units, rather than implementing a stateful mechanism from output to input units. More precisely, Todd criticises the interpretability of traditional RNNs by remarking that "*as the hidden units typically compute some complicated, often uninterpretable function of their inputs, the memory kept in the context units will likely also be uninterpretable*".

Limitations Although the contribution is dated at the time of writing, the idea of fully memorising a limited collection of pieces is drastically different from all the current approaches,

and corroborates the need of explicit memory in music modelling. Nonetheless, learning musical features from only 4 short pieces is considerably difficult, as the model is intentionally design to overfit such sequences. In this scenario, few-shot learning approaches seem to be better candidates for this challenge. Another concern, regarding the asymmetric music representations, is the overhead of the plan inputs, which requires 1 event per musical sequences to learn. This clearly jeopardises the use of this method on large training sets.

Learning musical sequences with a language model

In contrast to the previous approaches, B. L. Sturm et al. (2016) introduces an alternative perspective to model monophonic music and learn long-term metrical structure. More precisely, his approach exploits the format of symbolic music represented in the ABC notation, so that a language model can be learned from such musical sequences expressed in textual format.

The author investigates the use of a *character-level* and a *word-level* language model for monophonic modelling. Both these models are based on the LSTM, but operate over different vocabularies and are trained diversely. As the name suggests, the *char-rnn* operates at the most granular level – the character level, and is trained on a continuous text file. In other words, it learns a *language model* over ABC characters. The *folk-rnn*, instead, operates over a vocabulary of transcription tokens, which are notation-dependent, and is trained on individual transcriptions. A valid transcription begins with a special start symbol, followed by a time signature token, a key token, and then a sequence of ABC tokens of 4 different types.

Music representation Musical sequences are encoded in ABC notation, where both the properties of a note, i.e. pitch and duration, are expressed in textual form. Time is implicitly represented by the temporal succession of ABC tokens. The training data comes from a weekly dump of an online platform used for sharing and discussing music played in *traditional music sessions*, where the prevalent genres are thus Celtic and Morris.

Model Each recurrent network has three hidden layers, each with 512 LSTM blocks, and a number of input and output units equal to the number of characters or tokens in their relative vocabulary. Musical events – characters for the *char-rnn* and ABC tokens for the *folk-rnn*, are locally encoded as one-hot vectors. A final *softmax* layer considers the output of each network as a probability distribution over its vocabulary.

Evaluation Although B. L. Sturm et al. did not provide a quantitative evaluation of the music models, an extensive qualitative analysis of their compositions was performed. This included a comparison of the descriptive statistics of some generations with those of the training

Authors	Architecture	Dataset	Representations	Prior knowledge	Prediction eval.	Generation eval.	Code
Mozer	RNN	custom (10 pieces)	custom enriched	representations	yes	discussion	no
Franklin	LSTM RNN	custom	custom enriched	representations	yes	discussion	no
Eck & Schmidhuber	LSTM RNN	custom blues dataset	piano-roll	input conditioning	no	discussion	no
Eck & Lapamle	LSTM RNN	session.org	piano-roll	input conditioning	no	discussion	no
Lattner	GAE + RNN	EFSC	Humdrum (textual, no duration)	-	cross entropy, accuracy	no	no
Todd	RNN (Jordan)	custom (4 pieces)	piano-roll	-	no	discussion	no
Sturm	LSTM RNN	session.org	ABC (text)	-	no	statistical analysis, human expert	Theano

Table 4.1: Summary table of monophonic melody models

data; the analysis thus provided a weak measure of plagiarism for the synthetic sequences. To evaluate specific musical properties, folk musicians from an online community (`session.org`) were asked to manually inspect a set of *char-rnn*'s compositions. In addition, the latter model was also evaluated as a music continuation system, providing computationally creative support.

4.2.2 Polyphonic melody modelling

In this part, an overview of the most relevant contributions for polyphonic music modelling is presented, where the symbolic musical sequences under study express melodies played by a polyphonic instrument/voice (e.g. piano, guitar). Considering the incremental complexity of music modelling problems, all the works discussed here can also serve the monophonic case.

As illustrated in Figure 4.7, the task inherits all the technical challenges of the monophonic settings, and introduces the problem of learning note simultaneities. Overall, polyphonic models need to deal with bi-directional dependencies, spanning across the horizontal (temporal) and the vertical (harmonic) dimensions of music (c.f. Section 4.1.2). Similarly to the previous section, the literature review categorises polyphonic models according to the nature of their methodology, and relates these groups with the technical issues they intend to address (Figure 4.7). The resulting taxonomy is presented as follows.

Composite architectures. This group includes methods using a combination of two or more neural networks to model bi-directional dependencies, based on a subdivision of tasks: one network learns harmonic dependencies (the vertical dimension), and another is specialised in learning temporal dependencies (the horizontal dimension); these models are typically tied together and trained jointly. Two specific approaches are outlined as follows.

- *Recurrent probability estimators*. This effective yet computationally intensive methodology consists in exploiting a recurrent network to temporally adapt the parameters of

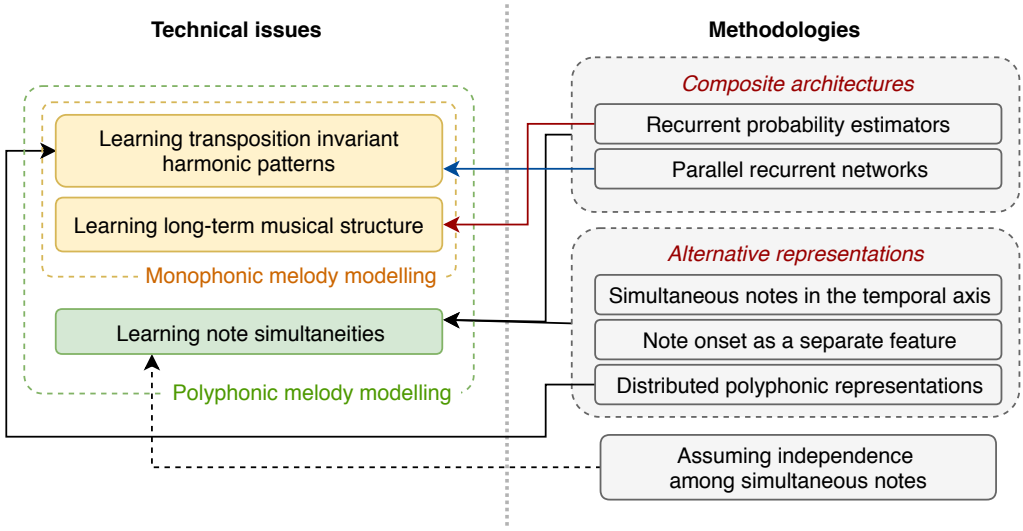


Figure 4.7: Technical issues vs current methodologies in polyphonic melody modelling

a feed-forward network while processing a musical sequence. More precisely, a probability density estimator models the vertical dimension of polyphonic music, whereas an RNN provides a dynamic parameterization of the probability estimator to account for the evolving temporal context (the horizontal dimension). Models adopting this approach are based on the following works: (Boulanger-Lewandowski et al., 2012; Goel, Vohra, & Sahoo, 2014; Lyu, Wu, & Zhu, 2015; Vohra, Goel, & Sahoo, 2015).

- *Parallel recurrent network*. This methodology, mainly based on Johnson's work, considers the concurrent use of two recurrent networks that process the temporal and the harmonic dimensions respectively. The novelty of this approach lies in the way these networks are connected/tied with each other. The resulting architecture is also designed to encode certain musical properties facilitating the model to learn transposition-invariant features.

The main drawbacks of these methodologies are the expensive sampling procedures (e.g. contrastive divergence, Gibbs sampling) required by recurrent probability estimators to predict chords, and the high degree of conditioning in parallel recurrent networks, requiring metrical and harmonic information as additional inputs to the music model.

Using alternative musical representations. In contrast to the previous group, these methods do not model note simultaneities as the joint probability of pitches, but they simplify the problem by leveraging alternative music representations. This can be done at the encoding level, during data pre-processing, or using the first layers of the network to learn numerical representations of polyphonic events (alias music embeddings) which are further processed for modelling. The main methodologies in this group are outlined as follows.

- *Polyphonic elements in the time axis*. As introduced in Section 4.1.2, the simplification is operated at the encoding level, with polyphonic events unrolled in the temporal dimension. This allows to process chords sequentially as special subsequences of their constituent pitches (Liang, Gotham, Johnson, & Shotton, 2017; Walder, 2016). For instance, a C major chord would be expressed as a sub-sequence containing C, E, G pitches, delimited by special (start-end chord) tokens. The scope of these delimiters is twofold: on one hand, they let the network interpret the enclosed pitches as simultaneous elements; on the other hand, they ensure metrical correctness, as expanding a chord on the temporal dimension should not affect the actual length of a bar.

  On the downside, considering that the temporal dimension already poses several challenges for music models (Section 4.1.2), this approach further exacerbates the problem of learning long-term musical structure, as sequences would inevitably be elongated.

- *Distributed representations*. These approaches aim at overcoming the limitations of local representations, e.g., the extensively used piano-rolls encodings. Distributed representations of symbolic music can be either obtained from a learned embedding matrix (A. Huang & Wu, 2016), similarly to word embeddings in natural language processing, or building upon musicologist methods that formally describe tonal properties (Chuan & Herremans, 2018). In addition, when providing prior knowledge of music to learn musically meaningful representations, models using distributed representations are generally more effective at learning transposition-invariant musical features.

- *Note offset and duration as separate features*. Another method circumventing the problem of note simultaneities at the representation level, consists in explicitly encoding note onsets. In this way, not only a note is represented by their pitch and duration, but also considering its time of occurrence (note onset). The latter can be encoded as the temporal offset from the last note or from the beginning of the musical sequence. Using such alternative approach for modelling the evolution of time in a sequence, chords are implicitly defined by notes temporally overlapping (Shin et al., 2017).

Simultaneous notes as independent events. The simplest approach to deal with polyphonic events is to disregard their harmonic dependencies. Neural networks adopting this approach use a sigmoid output layer, modelling an independent probability distribution per pitch (Brunner et al., 2017; Ycart et al., 2017). Hence, the occurrence of a particular pitch does not influence the probability of other pitches being active at the same time.

In sum, the contributions belonging to the first category address the polyphonic problem from a probabilistic perspective; the second category leverages the use of flat encodings, where polyphonic events are made sequential or represented in a latent space; finally, the last category

treats simultaneous pitches as statistically independent events. The remainder of this section presents and discusses the most relevant contributions in each category of the taxonomy.

Composite architectures

The architectures of the models belonging to this category are obtained from the combination of two or more artificial neural networks, which collaborate with each other in order to capture the bi-directional dependencies of polyphonic music. Despite their effectiveness, these models are generally criticised for their complexity and for the high number of parameters, which in turn call for large training sets and have propensity to over-fitting (Ycart et al., 2017).

Recurrent probability estimators: RNN-RBM and RNN-NADE Though a bit dated at the time of writing, the contribution of Boulanger-Lewandowski et al. (2012) is still the state of the art in modelling bidirectional dependencies from a probabilistic perspective.

As note simultaneities can be considered as correlated patterns, the authors observed that a traditional RNN architecture designed for a multi-class classification task would not conveniently model this high-dimensional sequential process. In fact, this would require enumerating all possible configurations of the variable to predict, which is considerably expensive from a practical viewpoint (Boulanger-Lewandowski et al., 2012). In light of this observation, their contribution leverages *energy-based models*, which can express the negative log-likelihood of a given configuration via an arbitrary energy function. This resulted in the introduction of a novel family of neural networks, combining an RNN with a restricted Boltzman machine (RBM).

The resulting architecture exploits the capabilities of an RBM to represent a pitch distribution at each time step, with its parameters being dynamically adapted by a recurrent network. In other words, the behaviour of a probability density estimator, originally conceived for modelling non-sequential data, can be made time-dependent if a set of its parameters are let free to adapt while processing a high-dimensional sequence. This approach was initially investigated by Sutskever, Hinton, and Taylor (2009) for the recurrent temporal RBM (RTRBM), and further generalised by Boulanger-Lewandowski et al. (2012) with the introduction of the RNN-RBM.

To better explain this central contribution in the general context of high-dimensional sequence modelling, let us briefly introduce the concepts of RBM, RTRBM and RNN-RBM.

Restricted Boltzman machine (RBM) – an energy-based model where the joint probability of a given configuration of the input vector $\mathbf{v}$ and the hidden vector $\mathbf{h}$ is expressed as:

$$P(\mathbf{v},\mathbf{h}) = \exp(-\mathbf{b}_v^\top \mathbf{v} - \mathbf{b}_h^\top \mathbf{h} - \mathbf{h}^\top \mathbf{W}\mathbf{v})/Z \tag{4.3}$$

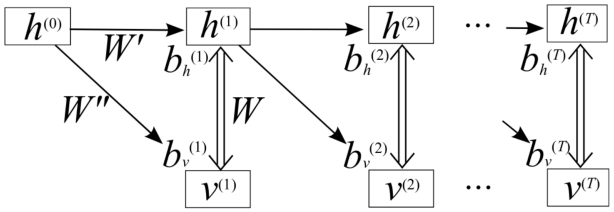


Figure 4.8: The RTRBM. Taken from (Boulanger-Lewandowski et al., 2012).

where $\mathbf{b}_v, \mathbf{b}_h$ and $\mathbf{W}$ are the parameters of the model and Z is the usually intractable partition function. Therefore, considering the restricted nature of the Boltzmann machine, when the visible vector $\mathbf{v}$ is provided in the input layer, the hidden units h_i are conditionally independent of one another, and viceversa:

$$P(h_i = 1|\mathbf{v}) = \sigma(\mathbf{b}_h + \mathbf{W}\mathbf{v})_i \tag{4.4}$$

$$P(v_j = 1|\mathbf{h}) = \sigma(\mathbf{b}_v + \mathbf{W}^\top \mathbf{h})_j \tag{4.5}$$

where σ denotes the element-wise sigmoid function and the indices i, j are used to refer to single elements of the vectors. The marginalised probability of $\mathbf{v}$ is related to the concept of free-energy $F(\mathbf{v})$ by $P(\mathbf{v}) = e^{-F(\mathbf{v})}/Z$ where:

$$F(\mathbf{v}) = -\mathbf{b}_v^\top \mathbf{v} - \sum_i \log(1 + e^{\mathbf{b}_h + \mathbf{W}\mathbf{v}})_i \tag{4.6}$$

Therefore, inference in an RBM consists of sampling h_i given $\mathbf{v}$ (or v_j given $\mathbf{h}$) according to their conditional Bernoulli distribution defined by Equations 4.4 and 4.5. Sampling $\mathbf{v}$ from the RBM can be done using block Gibbs sampling, i.e. by performing k alternating steps of sampling $\mathbf{h}$ from $\mathbf{v}$ and $\mathbf{v}$ from $\mathbf{h}$ (usually denoted as $\mathbf{h}|\mathbf{v}$ and $\mathbf{v}|\mathbf{h}$ respectively).

Recurrent temporal RBM (RTRBM) – a recurrent version of the RBM, or a sequence of conditional RBMs whose parameters $\mathbf{b}_v^{(t)}, \mathbf{b}_h^{(t)}, \mathbf{W}^{(t)}$ are time-dependent. In particular, these parameters depend on the sequence history (in terms of past elements) at time t, denoted as $\mathcal{A}^{(t)} = \{\mathbf{v}^{(\tau)}, \hat{\mathbf{h}}^{(\tau)} | \tau < t\}$ and where $\hat{\mathbf{h}}^{(\tau)}$ is the mean-field value of $\mathbf{h}^{(\tau)}$. Formally, the RTRBM models the following joint probability distribution:

$$P(\{\mathbf{v}^{(t)}, \mathbf{h}^{(t)}\}) = \prod_{t=1}^{T} P(\mathbf{v}^{(t)}, \mathbf{h}^{(t)} \mid \mathcal{A}^{(t)}), \tag{4.7}$$

where $P(\mathbf{v}^{(t)}, \mathbf{h}^{(t)} \mid \mathcal{A}^{(t)})$ can be considered as the joint probability of the t-th RBM whose

bias parameters depend on $\hat{\mathbf{h}}^{(t-1)}$, according to the equations reported below.

$$\mathbf{b}_h^{(t)} = \mathbf{b}_h + \mathbf{W}'\hat{\mathbf{h}}^{(t-1)} \tag{4.8}$$

$$\mathbf{b}_v^{(t)} = \mathbf{b}_v + \mathbf{W}''\hat{\mathbf{h}}^{(t-1)} \tag{4.9}$$

A recurrent temporal RBM has thus 6 parameters, i.e. $\mathbf{W}, \mathbf{b}_v, \mathbf{b}_h, \mathbf{W}', \mathbf{W}'', \hat{\mathbf{h}}^{(0)}$. It is also important to remark that, whereas the hidden units $\mathbf{h}^{(t)}$ are binary during inference and sampling, it is the *mean-field* vector value $\hat{\mathbf{h}}^{(t)}$ that is transmitted to its successors.

$$\hat{\mathbf{h}}^{(t)} = \sigma(\mathbf{W}\mathbf{v}^{(t)} + \mathbf{b}_h^{(t)}) = \sigma(\mathbf{W}\mathbf{v}^{(t)} + \mathbf{W}'\,\hat{\mathbf{h}}^{(t-1)} + \mathbf{b}_h) \tag{4.10}$$

As observed by Boulanger-Lewandowski et al., the equation above can be interpreted as a single-layer recurrent neural network with hidden units $\hat{\mathbf{h}}^{(t)}$.

RNN-RBM – a generalisation of the RTRBM providing more flexibility to the hidden layer of the RBM. This contribution is motivated by the observation that a RTRBM can be considered as a sequence of RBM whose parameters are the output of a deterministic RNN, with the major constraint that the hidden units should model the conditional distributions as well as conveying temporal information. Therefore, to relieve the hidden states from such a twofold task and focus on the distributions, Boulanger-Lewandowski et al. (2012) combined a full RNN, with separate hidden units $\hat{\mathbf{h}}^{(t)}$, with the RTRBM graphical model. The RNN-RBM thus defines the joint probability distribution as done by the RTRB in Equation 4.7, with the exception that $\hat{\mathbf{h}}^{(t)}$ is arbitrarily controlled by a vanilla RNN as follows:

$$\hat{\mathbf{h}}^{(t)} = \sigma(\mathbf{W}_2\mathbf{v}^{(t)} + \mathbf{W}_3\,\hat{\mathbf{h}}^{(t-1)} + \mathbf{b}_{\hat{h}}). \tag{4.11}$$

The resulting generalised RNN-RBM architecture has nine parameters: those of the RBM, i.e. $\mathbf{W}, \mathbf{b}_v, \mathbf{b}_h, \mathbf{W}', \mathbf{W}'', \hat{\mathbf{h}}^{(0)}$, and the parameters of the RNN, i.e. $\mathbf{W}_2, \mathbf{W}_3, \mathbf{b}_{\hat{h}}$. It can easily be showed that a single-layer RNN-RBM is a generalisation of the RTRBM by observing that the former architecture reduces to the latter if $\mathbf{W}_2 = \mathbf{W}$, $\mathbf{W}_3 = \mathbf{W}'$ and $\mathbf{b}_{\hat{h}} = \mathbf{b}_h$.

Figures 4.8 and 4.9 compare the graphical structures of the RTRBM and the RNN-RBM; where single arrows represent a deterministic function, and double arrows represent the stochastic hidden-visible connections of an RBM. For more technical details on the training procedure, the initialisation strategies, and the adaptation of the BPTT algorithm for such architectures, we refer to (Boulanger-Lewandowski et al., 2012).

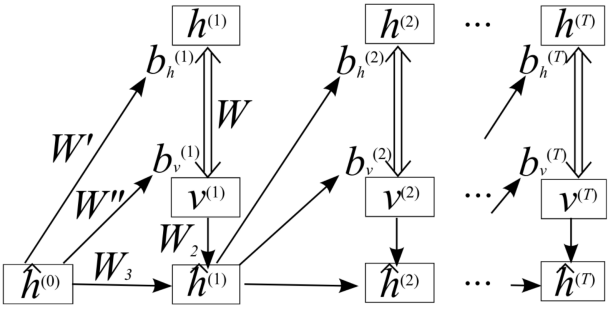


Figure 4.9: RNN-RBM graphical model. Taken from (Boulanger-Lewandowski et al., 2012).

Neural autoregressive distribution estimator (NADE). Considering that the RBM has an intractable gradient to compute (requiring estimation via contrastive divergence or Gibbs sampling), the authors also investigated replacing the RBM with NADE (Uria, Côté, Gregor, Murray, & Larochelle, 2016). More precisely, NADE provides a tractable model for learning the joint distribution of high-dimensional variables. Denoting the set of pitches that can be played together as $\mathbf{v} = [v1, v2, \ldots, v_n]$, NADE approximates the joint probability of the variables contained in v using a series of conditional distributions:

$$p(\mathbf{v}) = \prod_{i=1}^{|\mathbf{v}|} p(v_i|\mathbf{v}_{<i}), \tag{4.12}$$

where each conditional distribution is defined according to the following equations:

$$\mathbf{h}_i = \sigma(\mathbf{b}_h + W_{:,<i}\mathbf{v}_{<i}) \tag{4.13}$$

$$p(v_i = 1|\mathbf{v}_{<i}) = \sigma(\mathbf{b}_{v_i} + V_{i,:}\mathbf{h}_i) \tag{4.14}$$

$$p(v_i = 0|\mathbf{v}_{<i}) = 1 - p(v_i = 1|\mathbf{v}_{<i}) \tag{4.15}$$

where $\mathbf{b}_v, \mathbf{b}_h$ are bias vectors and W and V are weight matrices, $\mathbf{v}_{<i}$ denotes the vector composed of the first $i-1$ elements of $\mathbf{v}$, $W_{:,<i}$ denotes the matrix composed of all rows and the first $i-1$ columns of W, and $V_{i,:}$ denotes the i-th row of V. Analogously to the RNN-RBM, the bias parameters of an **RNN-NADE** are updated at each time step using the hidden activations $\hat{\mathbf{h}}$ of an RNN. Under the distribution obtained assuming ordered dependencies in $\mathbf{v}$, the loss has a tractable gradient, hence no estimation method is needed.

The RNN-RBM was tested on four polyphonic datasets: *Piano-midi.de* (classical piano MIDI archive), *Nottingham* (folk tunes in ABC notation), *MuseData* (orchestral piano music) and *JSB chorales* (four-part harmonised chorales). Each dataset contains at least 7 hours of

polyphonic music, where the degree of polyphony varies from 0 to 15 simultaneous notes, with an average of 3.9 notes per time step. Musical sequences are encoded as piano-rolls quantised to the 4th note and covering the grand piano's pitch range (88 pitches, from A0 to C8).

To evaluate the predictive performance of the RNN-RBM polyphonic model, the authors proposed a comparison with other baseline approaches including n-grams, Gaussian mixture models combined with hidden Markov models, multi layer perceptrons, and other relevant works in the literature (Allan & Williams, 2005; Lavrenko & Pickens, 2003). Their results demonstrate that the RNN-RBM outperformed all the other models, both in terms of *log-likelihood* and *frame-level accuracy* (Bay, Ehmann, & Downie, 2009). Notably, using a NADE instead of the RBM comes at the cost of slightly decreased prediction performance, which is still negligible given that the former model is easier to train thanks to its tractable gradient.

The RNN-RBM was also combined with an acoustic model for the task of automatic music transcription (AMT), and was demonstrated to improve the performance of the resulting system. The model was also tested as part of an end-to-end neural-network for multi-pitch detection in piano music, coupled with a variety of acoustic models (Sigtia, Benetos, & Dixon, 2016).

To conclude, a qualitative evaluation was also carried out. This analysis pointed out that music generated by the RNN-RBM model was considered pleasant, adherent to basic harmonic rules and with local temporal coherence. Nevertheless, Boulanger-Lewandowski et al. explicitly mentioned that long-term structure was not captured by the model.

Recurrent probability estimators: LSTM-RTRBM To address the limited long-term capabilities of the RNN-RBM, Lyu et al. (2015) investigated the use of an LSTM recurrent network with the RTRBM architecture Sutskever et al. (2009). More precisely, the authors use bypass connections from LSTM units to the hidden units of the RTRBM, which are already includes connections over time. In fact, the recurrent temporal connections that characterise the RTRBM architecture are still present in the proposed model. In doing so, the recurrent hidden units of the RTRBM are said to capture short-term musical structure, whereas LSTM units serve as long-term memory. Differently from the RNN-RBM, there are two channels for temporal information: the direct connection between the conditional RBM at each time step, and the connection from the recurrent LSTM units of previous time steps (Lyu et al., 2015).

Recurrent probability estimators: RNN-DBN In a similar vein to the previous contribution, Goel et al. (2014) extended the RNN-RBM architecture (Boulanger-Lewandowski et al., 2012) with a more complex probability estimator for modelling the "vertical dimension" of polyphonic music. In particular, they combined the sequence modelling capabilities of the RNN with the high dimensional data modelling of deep belief networks (DBNs). In short, this is done by

introducing an additional hidden layer in the RBM architecture. To better introduce the RNN-DBN model, a concise description of deep belief networks is provided below.

A **deep belief network** (DBN) is a generative graphical model that learns to extract a deep hierarchical representation of the training data (Hinton, Osindero, & Teh, 2006). Described as a stack of RBMs trained greedily, a DBN is expected to model the joint probability distribution between the observed vector $\mathbf{v}$ and the l hidden layers h^k as follows:

$$P(\mathbf{v}, \mathbf{h}^1, \ldots, \mathbf{h}^l) = \Big(\prod_{k=0}^{l-2} P(\mathbf{h}^k | \mathbf{h}^{k+1}) \Big) P(\mathbf{h}^{l-1} | \mathbf{h}^l), \tag{4.16}$$

where $\mathbf{v} = \mathbf{h}^0$, $P(\mathbf{h}^k | \mathbf{h}^{k+1})$ is a conditional distribution for the visible units conditioned on the hidden units of the RBM at the successive level k; and $P(\mathbf{h}^{l-1} | \mathbf{h}^l)$ is the visible-hidden joint distribution in the top-level RBM. According to Bengio et al. (2009), the principle of greedy layer-wise unsupervised training can indeed be applied to deep belief networks with RBMs as the building blocks for each layer.

The **RNN-DBN** architecture replaces the RBM portion of the RNN-RBM model with a DBN in order to lift the limited expressiveness of a single RBM layer. Though an arbitrary number of hidden layers can be used in principle, Goel et al. included two hidden layers in the DBN. Following the design of the RNN-RBM, only the bias vectors of the two hidden layers are controlled (and updated at each time step) by the hidden state of the recurrent network at the previous time step. For instance, the bias vector of the second hidden layer is defined according to the following equation:

$$\mathbf{b}_{h_2}^{(t)} = \mathbf{b}_{h_2} + \mathbf{W}_{uh_2} \mathbf{u}^{(t-1)}, \tag{4.17}$$

where $\mathbf{u}^{(t-1)}$ denotes the hidden state of the RNN at time step $t-1$. As can be observed, $\mathbf{u}^{(t-1)}$ corresponds to $\hat{\mathbf{h}}^{(t-1)}$ in the work of Boulanger-Lewandowski et al. (2012). The graphical model of a RNN-DBN with two hidden layers as in Figure 4.10.

The authors evaluated the predictive capabilities of the proposed architecture on the same datasets used in the experiments for the RNN-RBM (Boulanger-Lewandowski et al., 2012). Their results indicate that an RNN-DBN with two hidden layers achieves comparable results to the RNN-RBM. Hence, it is still not clear whether the adoption of a DBN is preferable to a simple RBM, for the purpose of capturing harmonic dependencies among simultaneous notes.

To conclude, Vohra et al. (2015) investigated the use of an LSTM to replace the vanilla RNN in the RNN-DBN architecture. The resulting model – the LSTM-DBN, thus complements

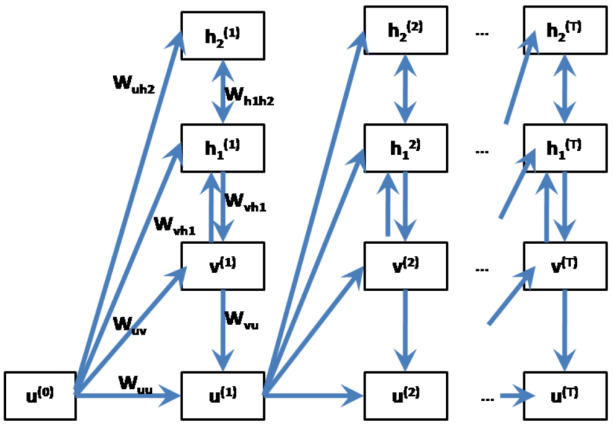


Figure 4.10: An RNN-DBN with 2 hidden layers. Taken from (Goel et al., 2014).

the more general formulation of Goel et al. (2014) with the increased memory capabilities demonstrated in (Lyu et al., 2015), and can be considered as a state of the art polyphonic model.

Parallel recurrent networks The category includes two different composite recurrent architectures, inspired by convolution, that attain transposition invariance and produce joint-probability distributions for polyphonic events over a musical sequence Johnson (2017).

This line of work is motivated by the way harmonic dependencies can be expressed in relative terms, describing another invariant property of music. For instance, assuming local pitch representations per semitone (half-step increases), a major chord can be encoded as:

...00100010100...

As a major chord with a certain tonic consists of its major third and perfect fifth (+4 and +6 semitones from the tonic, respectively), the corresponding pattern remains unchanged regardless to where it appears in a feature vector – a property known as *translation invariance*. In fact, all the following example representations would still represent major chords:

100010100000, 001000101000, 000001000101

.

However, a probability distribution estimator such as NADE or RBM, would need to learn each transposed representation separately. Instead of enumerating all possible major chords, a model should account for their cross-correlation and learn transposition invariant patterns. More formally, if we have a vector of notes $\mathbf{v}^{(t)}$ at time step t and a candidate vector of notes $\hat{\mathbf{v}}^{(t+1)}$ for the next time step, and we construct shifted vectors $\mathbf{w}^{(t)}$ and $\hat{\mathbf{w}}^{(t+1)}$ according to some offset δ, such that $\mathbf{w}_i^{(t)} = \mathbf{v}_{i+\delta}^{(t)}$ and $\hat{\mathbf{w}}_i^{(t+1)} = \hat{\mathbf{v}}_{i+\delta}^{(t+1)}$, then we want the output of our model to

satisfy the following condition:

$$p(\hat{\mathbf{w}}^{(t+1)}|\mathbf{w}^{(t)}) = p(\hat{\mathbf{v}}^{(t+1)}|\mathbf{v}^{(t)}). \tag{4.18}$$

The address this issue, Johnson proposed two different recurrent architectures for polyphonic music modelling. In both these architectures, transposition invariance is considered analogously to the definition of positional invariance in convolutional neural networks.

Tied-parallel LSTM-NADE. The idea consists in partitioning the whole architecture of the model into a set of tied parallel networks. Each network instance is responsible for a single note, and has tied weights with every other network instance. Since each instance uses the same procedure to compute its output, shifting the inputs up by some amount δ would shift the outputs accordingly, thus ensuring translation invariance.

Each network instance, specialising in handling a specific note, is implemented as a RNN-NADE (Boulanger-Lewandowski et al., 2012). Given a note vector $\mathbf{v}^{(t)}$ as input, each network instance n receives two input features: a local window $\mathbf{w}^{(n,t)}$, and a set of bins $\mathbf{z}^{(n,t)}$. The local window contains a slice of the note vector $\mathbf{v}^{(t)}$ giving a span of one octave above and below the note, i.e. $\mathbf{w}_i^{(n,t)} = \mathbf{v}_{n-13+i}^{(t)}$ where $1 \leq i \leq 25$. The content of each bin $\mathbf{z}_i^{(n,t)}$ is the number of notes played at time t, and at the offset i from note n across all octaves. In addition, since music is said to be mostly translation-invariant for small shifts, the input also includes the (absolute) MIDI number of the corresponding pitch (differences between high and low notes deserve special treatment). All these inputs are concatenated in a single vector, then fed to the RNN-NADEs, where each RNN is implemented with an LSTM.

Bi-axial LSTMs. This second architecture is motivated by the fact that the previous model uses windowed and binned summaries of note output, which in turn prevents the model from learning any dependencies that extends past the size of the window. To overcome this limitation, Johnson replaced the NADE portion of the model with another LSTM having recurrent connections along the note axis. The resulting architecture is based on a *bi-axial* configuration (which should not be confounded with bidirectional RNNs), where an LSTM processes music along the time axis while another "walks through" along the note axis.

More precisely, at each note-step, the note-axis LSTM layers receive a concatenation of two inputs: the activations of the final time-axis LSTM layer for a note, as well as the final output of the network for the previous note. The final activations of the note-axis LSTM is then be transformed into a probability $p^{(t,n)}(v_n = 1|\mathbf{v}_{<n})$ using softmax activation. As can be observed, just as each note has a corresponding tied-weight time-axis LSTM network responsible for modelling temporal relationships for that particular note, each time step has

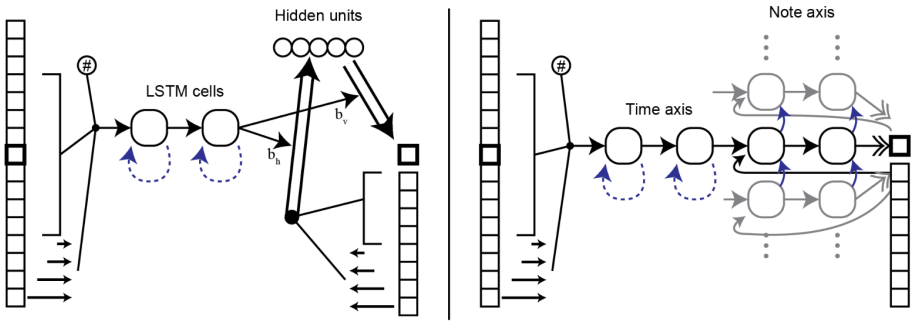


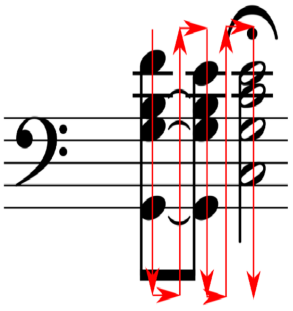
Figure 4.11: Left: Illustration of a network instance in the tied-parallel LSTM-NADE architecture, responsible for outputting the probability for the note indicated by the thick-outlined box. Right: an example of bi-axial LSTM, showing a single instance of the time-axis network, and three note-steps of the note-axis network. For each network, a window of the note's vicinity, bins, and the MIDI number of the current note are concatenated. Concatenations are indicated by lines connected by a solid black circle. Dashed arrows represent time-delayed connections, blue arrows represent recurrent connections, thick double-line-arrows represent the adapted NADE estimator, and double-headed arrows indicate a sampling operation.

a corresponding tie-weight note axis LSTM network responsible for modelling the joint distribution of notes in that single time step. Therefore, sequentially running this network for each note in a time step provides the full conditional distribution for that step.

Both these architectures were trained on four popular datasets – *JSB Chorales*, *MuseData*, *Nottingham*, and *Piano-midi.de*, and compared with the RNN-RBM, RNN-NADE (Boulanger-Lewandowski et al., 2012) and a DBN-LSTM based on (Goel et al., 2014). According to their results, the bi-axial LSTMs outperformed the tied-parallel LSTM-NADE, but not the DBN-LSTM Vohra et al. (2015). The performance gap with the DBN-LSTM is potentially attributable to the ability of deep belief networks to capture a richer joint distribution at each step.

Another quantitative evaluation tested whether parallel networks can effectively learn transposition invariant features. This was done by comparing the log-likelihood of the models' predictions on transposed versions of the test set, which is expected to be similar to that of the models' predictions on the original set. Results confirmed that both the parallel networks learn musical regularities that generalise to music in multiple keys, and are not sensitive to transpositions of the input – a property not directly shown by LSTM-NADE (Johnson, 2017).

In conclusion, the author also performed a qualitative evaluation on compositions generated with the biaxial LSTM. For this task, several architectural adaptations were made. To facilitate the network in learning metrical regularities, the model's input was extended with a vector providing the relative position in a refcerence 4/4 measure. In addition, to overcome the issue of consecutive note repetitions note in piano-roll representations, another binary dimension was added to the note vector $\mathbf{v}$ to distinguish a note being repeated from the same being sustained.



(a) Three chords in music notation, where red arrows indicate the order in which notes are encoded.

```
START
(65, False)
(59, False)
(55, False)
(43, False)
|||
(64, False)
(59, True)
(55, True)
(43, True)
|||
(.)
(64, False)
(60, False)
(55, False)
(48, False)
|||
END
```

(b) The corresponding sequential encoding using 8th-note quantisation (broken over three columns only for illustration). Each line within a column corresponds to an individual token in the encoded sequence. ||| delimit frames and (.) indicates a fermata in the corresponding frame.

Figure 4.12: An example of music encodings in BachBot, representing three chords ending with a fermata. Taken from (Liang et al., 2017).

Given that Johnson's parallel networks are two of the very few models that can effectively learn transposition invariant features, without any intervention at the representation level, and note simultaneities, this line of work represents a central contribution in the literature.

Alternative representations: polyphonic elements in the time axis

The general approach characterising the contributions in this category is based on breaking chords into their constituent pitches, to represent them on the temporal axis as special subsequences. To preserve the length of measures, and ensure that a model can always distinguish polyphonic sub-sequences, special delimiters are usually employed for delimiting the boundaries of the translated simultaneous notes.

A representative example in this category is BachBot (Liang et al., 2017), introducing a new sequential encoding scheme for polyphonic music and a model that can be used for both composition and harmonisation. Even though this work aims at modelling Bach chorales – a form of composition for 4 distinct monophonic voices (i.e. *soprano*, *alto*, *tenor*, *bass*), this task can still be considered as a polyphonic modelling problem as all voices are in fact monophonic.

The proposed encoding represents scores with a quantization to the 16th note. Each frame (time step) includes a $\langle Pitch, Tie \rangle$ tuple for each voice in a chorale – soprano, alto, tenor, and bass, where $Pitch \in \{0, 1, \ldots, 127\}$ encodes the MIDI number of a note and $Tie \in \{True, False\}$ specifies whether a note is tied with a note at the same pitch from the previous frame, or is articulated at the current time step. In this context, simultaneous notes appearing in the same frame (a maximum of four), are considered as a sub-sequence per se and are delimited by a special character – the ||| symbol. In other words, all the pitches within the same frame are "unrolled" on the temporal axis, so that they can be processed sequentially. As can be

noticed from the example in Figure 4.12, notes within a frame are processed in descending order according to their MIDI number and neglecting crossing voices (from soprano to bass).

To address long-term dependencies in musical sequences, Liang et al. use a stack of three LSTM recurrent networks, each with hidden size of 256 units per cell. The model first embeds a sequence element into a 32-dimensional vector space. Next, the note embedding is passed to 3 LSTM recurrent layers followed by batch normalisation and dropout. To conclude, a fully-connected and a softmax layer are used to yield the distribution of the next token.

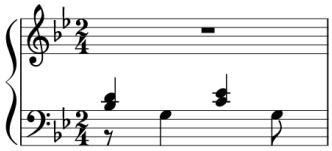
A quantitative evaluation of the predictive performance of BachBot was reported in (Liang et al., 2017). In addition to the average cross-entropy loss on test sequences, the authors also provided a *token error rate* (TER), intended as the percentage of errors in individual token prediction, as well as a *frame error rate* (FER), defined as the percentage of errors in frame predictions, where any mistakenly predicted token within a frame counts as a frame error. To evaluate the generative capabilities of BachBot, the authors conducted one of the largest musical discrimination tests ever carried out on a collection of real and model-generated compositions. For this purpose, an online platform was implemented to give anyone access to the musical discrimination test, rather than using paid services offering monetary incentives for participation.

Another work implementing the current methodology is that of Walder (2016). With the introduction of alternative representations reducing polyphonic music to univariate categorical sequences, the author applies state of the art NLP techniques for the polyphonic music modelling task, without the need of using a probability density estimator, such as RBM and DBN.

In particular, Walder's method casts the polyphonic settings to the problem of modelling a density over multisets $\mathbf{c} \in \mathcal{C}$ with underlying set of elements $\mathcal{X} = \{1, 2, \ldots, d\}$. In this context, $\mathcal{X}$ denotes the set of d note pitches (e.g. $|\mathcal{X}| = 88$ for a gran piano), and $\mathbf{c}$ is an unordered set of note pitches defining a chord. Since $\mathcal{X}$ contains ordinal elements, a bijective function can be defined as $f : \mathcal{C} \to O$, where O is the set of ordered tuples with underlying set of elements $\mathcal{X}$. Hence, for $\mathbf{o} \in O$ s.t. $\mathbf{o} = (o_1 \leq o_2 \leq \cdots \leq o_{|\mathbf{O}|})$ the problem of modelling simultaneous notes can be formulated as follows:

$$p(\mathbf{o}) = \prod_{i=1}^{|\mathbf{o}|} p(o_i | o_1, o_2, \ldots, o_{i-1}). \tag{4.19}$$

In other words, the task of learning the harmonic dependencies among simultaneous pitches is cast as another instance of the sequence modelling problem. Therefore, Walder's method consists in modelling each note sequentially, first ordered temporally and then, for notes played at the same time, in ascending order of pitch. As can be observed, this approach is similar to the previous one by Liang et al. (2017), but does not limit the degree of polyphony as done for Bach's chorales (4 voices). For this reason, Equation 4.19 provides a general framework to all



midi 63	0	0	0	0	0	0	1	1	1	0
midi 62	0	0	1	1	0	0	0	0	0	0
midi 60	0	0	0	0	0	1	1	1	1	0
midi 58	0	1	1	1	0	0	0	0	0	0
midi 55	0	0	0	1	1	1	1	0	1	0
t	0	0	0	0.5	1	1	1	1.5	1.5	2
Δt_{event}	1	1	1	0	1	1	0	0.5	0	0
Δt_{step}	0	0	0.5	0.5	0	0	0.5	0	0.5	0
ϵ_{onset}	0	1	0	0	0	1	0	0	0	0
ϵ_{offset}	0	0	0	0	1	0	0	1	0	1
target	58	62	55	-	60	63	-	55	-	-
lower bound	0	58	0	-	0	60	-	0	-	-

Figure 4.13: An example of Walder's representation. Taken from Walder (2016).

those models reducing polyphonic music sequences to monophonic melodies.

To implement this method, Walder designed a sophisticated encoding system which is illustrated in Figure 4.13. A musical fragment is unrolled into an input matrix (first ten rows), a target row, and a lower bound row. At each time step, an LSTM is presented with a column of the input, and has to predict the value in the target row – the MIDI number of the next note to be played. The MIDI rows of the input indicate which notes are active at a given time; in practice, considering the reduction of polyphonic music, only one note can turn on at a time. To facilitate the prediction task, the non-MIDI input rows (6 to 10) represent the rhythmic structure of the current note: Δt_{event} is the duration of the note being predicted; Δt_{step} is the time since the previous input column; ϵ_{onset} is 1 if and only if we are predicting in the same time as in the previous column; ϵ_{offset} is 1 if and only if we are turning notes off at the same time as in the previous column; and t represents the time step in the piece associated to the current column.

The author observed that the proposed note-based representation allows arbitrary timing information, and is not restricted to a uniform discretization of time as often done in the literature with piano-roll encodings. To summarise the main novelties of this approach:

- Notes composing chords are separated and moved to the temporal axis where they are presented in ascending pitch order (a note precedes another if it has lower pitch).

- The resulting musical sequence is processed note after note, although their relative duration (temporal offset) is only given as input but not predicted by the model.

Overall, the main limitation of this method is the lack of any rhythmic information in the predictions, as only the pitch of the next note is predicted at each time step. Therefore, note duration is provided as a conditioning signal at the input level to facilitate the task.

The predictive capabilities of the LSTM network trained on such representations were evaluated and compared with some of the composite architectures outlined in Section 4.2.2. Their

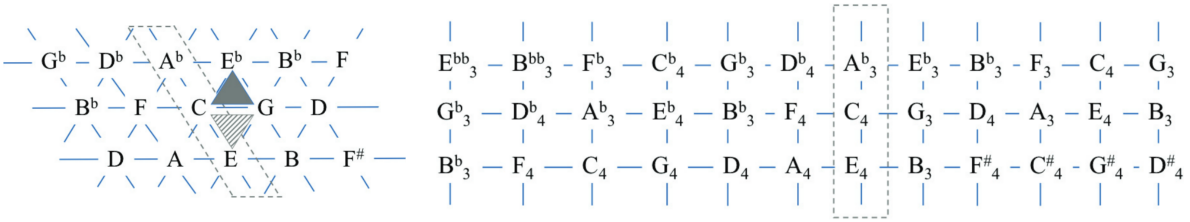


Figure 4.14: Example of a Tonnetz (*Left*) with a portion of the extended tonnetz matrix (*Right*). In the tonnetx matrix, pitch register is defined as the distance to the pitch from the center column highlighted in the dashed box. For instance, the register of G_3 next to C_4 in the second row is 3 instead of 4, as G_3 is closer to C_4 than G_4 in half steps. Taken from (A. Huang & Wu, 2016).

results indicate that the model can outperform the RNN-NADE (Boulanger-Lewandowski et al., 2012), but not the DBN-LSTM (Vohra et al., 2015) for polyphonic music modelling.

Approaches based on distributed representations

This part provides an overview of the most relevant works addressing the polyphonic problem by leveraging the use of distributed representations of musical notes. In fact, avoiding local representations frees the model from using sigmoidal output layers which would otherwise assume harmonic independence between simultaneous pitches. The following contributions borrow methods and tools from the fields of natural language processing and musicology to learn and obtain richer representations of musical elements, respectively.

The work by Chuan and Herremans (2018) proposes a novel method that translates each time slice of a polyphonic sequence into a 2-dimensional representation incorporating domain knowledge directly at the input level. Differently from the psychologically-based representations of Mozer (1994), these novel representations are based on *Tonnetz* – a musicologist visualisation method that emphasises the tonal relationships among pitches. To deal with these 2D representations at each time slice, an architecture combining the temporal abilities of LSTMs with a CNN autoencoder is proposed: the convolutional autoencoder captures musically meaningful relationships between pitches in the tonal space; the encoded Tonnetz sequence is then fed to the LSTM to model temporal dependencies, hence to predict the next time slice.

Tonnetz is a graphical representation used by music theorists to study tonality. As can be seen from Figure 4.14 (Left), each node in the tonnetz network represents one of the 12 pitch classes, where nodes on the same horizontal line follow the *circle-of-fifth* ordering: the adjacent right neighbour is the *perfect-fifth* and the adjacent left is the *perfect-fourth*. Three nodes forming a triangle denote a *triad*, and two triangles connected in the vertical direction are the *parallel major* and *minor triads* respectively (e.g. the upside-down triangle filled with diagonal lines is a C major triad, and the solid triangle on the top is a C minor triad).

As the size of the tonnetz mesh can be expanded recursively, so that a pitch class would appear in multiple places throughout the network, Chuan and Herremans unrolled the tonnetz into a 24-by-12 matrix, (Figure 4.14, right). Each node in the matrix represents a pitch instead of a pitch class (e.g. G_3, G_4 instead of G), so that the pitch register information is kept. The number of rows in the tonnetz matrix can be determined by the range of pitches of interest in a particular study. In the context of this work, 24 rows cover the pitch range from $C0$ to $C\#8$.

The proposed tonnetz representations provide a compact way for modelling simultaneous pitches. Each musical sequence is divided into time slices of 1 beat duration (e.g. a quarter note quantisation), so that pitches played at the same beat receive label 1 in the tonnetz matrix (0 otherwise). During pre-processing, all sequences are transposed to either C major or A minor, depending on the mode of its original key signature. In doing so, the harmonic role of a pitch is preserved (e.g. tonic, dominant), and represented in the same location in the tonnetz matrix. The resulting musical representations can thus be considered as a spatial version of piano-rolls encoding harmonic relationships between pitches (rather than their ordinal relationships only).

To process these representations, the proposed architecture is composed of two main parts.

1. A convolutional autoencoder (AE) allows the network to learn a compact (latent) representation of tonnetz matrices. An AE is generally used to learn a model that first encodes (compress) the input into a meaningful abstract representation – in a lower dimensional space – which in turn is used to decode (reconstruct) the original input as accurately as possible. In this context, the input is a tonnetz matrix $\mathbf{X}$ and the loss function is defined as the binary cross entropy among $\mathbf{X}$ and the encoded-decoded version $\mathbf{X}'$.

   The autoencoder used by Chuan and Herremans consists of two convolutional layers for the encoder part and one fully-connected layer for the decoder part. Each encoding layer has a convolution layer and a pooling layer. In the convolutional layer, each input neuron (corresponding to a node in the extended tonnetz matrix) is connected to a local receptive field, i.e. a 3×3 kernel specifically designed based on the number of pitches in a triad. A subsequent pooling layer produces a condensed version of the feature maps.

2. An LSTM receives the encoded representations at the current time step (Figure 4.15), and processes them sequentially in order to predict the tonnetz-based matrix at the next time step. The LSTM network has three hidden layers and uses standard LSTM cells.

The resulting architecture was trained and tested on the *MuseData* dataset for both the tasks of music modelling and generation. To quantify the contributions of each architectural component, the authors experimented with three configurations: no AE (LSTM only); training the whole network together (AE + LSTM); and pre-training the autoencoder, then freezing

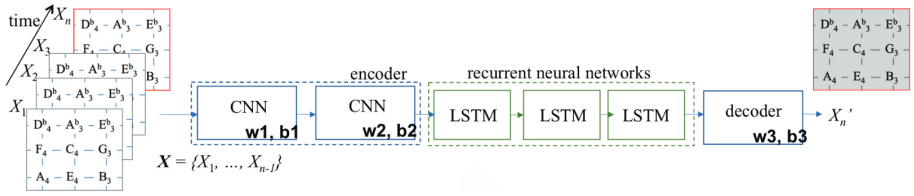


Figure 4.15: Illustration of the proposed architecture, taken from (Chuan & Herremans, 2018).

its weights when training with the LSTM. The obtained results revealed that the pre-trained configuration converges much quicker than the others, but all of them achieve the same loss.

To evaluate the effectiveness of the proposed representations, two model instances were trained on tonnetz and piano-roll representations, respectively. From their results, the predictive performance of these models, measured using cross entropy loss, precision, and recall, did not indicate any significant differences between them. Nonetheless, a comparison of the models' generations using computational metrics of musical tension – *cloud diameter* (a measure of dissonance), *cloud momentum* (a measure indicating changes in tonality), and *tensile strain* (a measure indicating tonal distance from the global key of the piece), demonstrated that tonnetz-based representations yield more stable tonal predictions. Moreover, the authors computed the compression ratio of musical sequences generated from both models. Given the current challenge of modelling and generating music with long-term structure, the compression ratio was said to provide insights on the ability of a model to generate repeated patterns.

Following the same line of thought, A. Huang and Wu (2016) investigated the impact of using two different symbolic music representations on the performance of a polyphonic model. More precisely, a 2-layer LSTM recurrent network was used separately in two different experiments where both MIDI and piano-roll representations were respectively evaluated.

In their MIDI experiment, each MIDI message was encoded as a single token in the sequence, whereas in the piano-roll experiment, each unique combination of pitches across all time steps was enumerated as a separate token. Both these methods allow processing polyphonic music, although the complexity of the problem is simplified at the encoding level: MIDI messages are indeed temporally ordered and come with a relative time stamp; enumerating all possible chords is another effective strategy as discussed in Section 4.1.1. In both cases, A. Huang and Wu used an embedding matrix mapping each token into a learned vector space. After mapping a sequence of tokens into a sequence of embedding vectors, the resulting representations are concatenated and autoregressively fed to the LSTM recurrent network.

Unfortunately, the authors did not report any quantitative evaluation, which could have lead to a better understanding on whether the use of different representations influenced the predictive performance of the LSTM model. Nevertheless, A. Huang and Wu tried to assess the quality

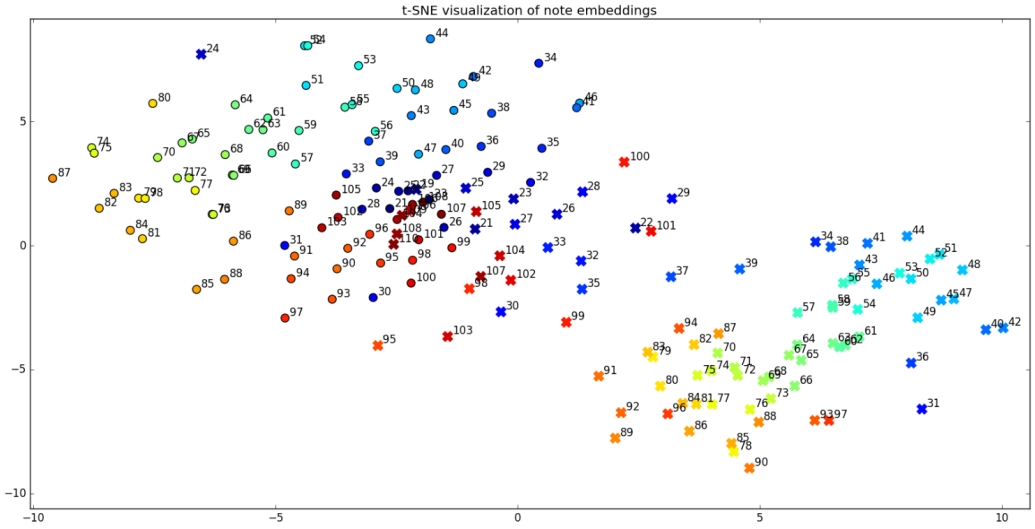


Figure 4.16: Visualisation of the note embeddings learned from MIDI events using a dimensionality reduction method (t-SNE). Taken from (A. Huang & Wu, 2016).

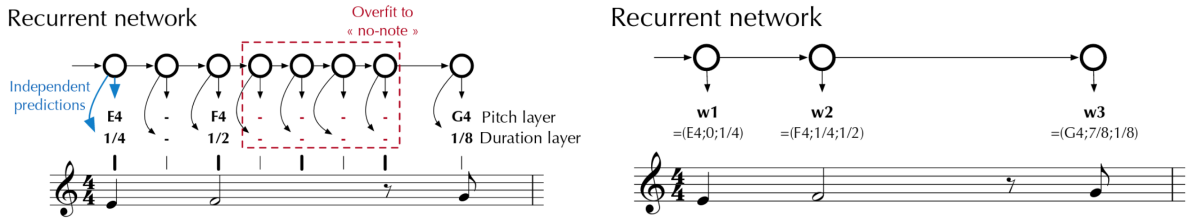
of the generated musical sequences by means of human involvement. In particular, they devised a blind experiment where 26 volunteers were asked to evaluate 3 generated compositions. Their results indicated that the models could generate music that is "*comparable in aesthetic quality*" to the musical sequences generated by the RNN-NADE (Boulanger-Lewandowski et al., 2012).

Note onset and duration as separate features

This group collects all methods representing musical sequences as a stream of notes (multimodal signals), so that models can process them note after note. In particular, each note is encoded according to its time of occurrence (alias, note onset), its duration and pitch, which are all represented as separate features. In doing so, simultaneous notes would trivially occur as "overlapping notes", without the need to process them together in the same time step.

Shin et al. (2017) implemented this approach by leveraging symbolic music encoded in MIDI format. The rationale behind their contribution comes from the observation that music in MIDI format can be considered as a "*discrete abstraction of musical sound*", analogously to the relationship among text and speech. As such, they proposed to treat notes and their distinctive properties as unique words, so that a note-level music model can be trained.

The authors specifically focused on pop songs, defined as relatively short compositions (mostly around 3-minute duration) with simple and memorable melodies that have relatively low structural complexity compared to other genres, such as classical music. As for the proposed representations, a pitch p_i with duration l_i located at time t_i from the start of the sequence is represented as a single token $n_i = (p_i, t_i, l_i)$. In this way, a melody is simply defined as



(a) Quantisation-based processing

(b) Timestep-independent representations

a sequence of note tokens $s_j = (n_o, \ldots, n_{m_j}) \in S$. The music model processing each note – implemented as an LSTM recurrent network, is also provided with two additional inputs as conditioning signals: (i) the current chord played by an accompaniment instrument (harmonic conditioning); (ii) the current section of the song, e.g. intro, chorus, verse (form conditioning).

The ability to represent note onset and duration as separate features frees the modeller from the need to perform time quantisation, either relative (metric-based) or absolute. This is somewhat analogous to character- and word-level language models, which can be compared to quantisation-based and note-level music models. In this context, quantisation-based approaches produce sequences with a substantial number of repeated frames (e.g. 16 frames to encode a quarter note, assuming 16th-based quantisation). According to Shin et al., a model statistically trained on those sequences would be more biased towards repeating previous frames – a modelling behaviour that is predominantly found in generated compositions. Note-level encodings, instead, have several desirable properties such as producing more compact sequences, and not having to consider silent intervals. The authors also observe that separating note pitch, position, and duration is more coherent with the process of music composition (Levitin, 2006).

To evaluate the proposed method, Shin et al. (2017) focused on the music generation capabilities of their model and did not assess its predictive performance. For this purpose, they performed two experiments on a crowdsourcing platform (Amazon Mechanical Turk) where human participants were asked to evaluate the models' compositions and compare these with music generated from (Chu, Urtasun, & Fidler, 2016) and (Jaques, Gu, Turner, & Eck, 2017).

From a critical perspective, conditioning a model on chords and structural information might not be viable in several cases, as symbolic music datasets providing both chords and structural annotations are relatively scarce. In their experiments, this was possible as such information was manually annotated by the authors on a training set that only contained 46 pop songs.

Modelling simultaneous notes as independent events

This last category closes the taxonomy by presenting a rather simplistic approach that consists in treating simultaneous notes as independent events to enumerate. Technically, this corresponds to using an output layer with sigmoid units – one for each pitch within the chosen range.

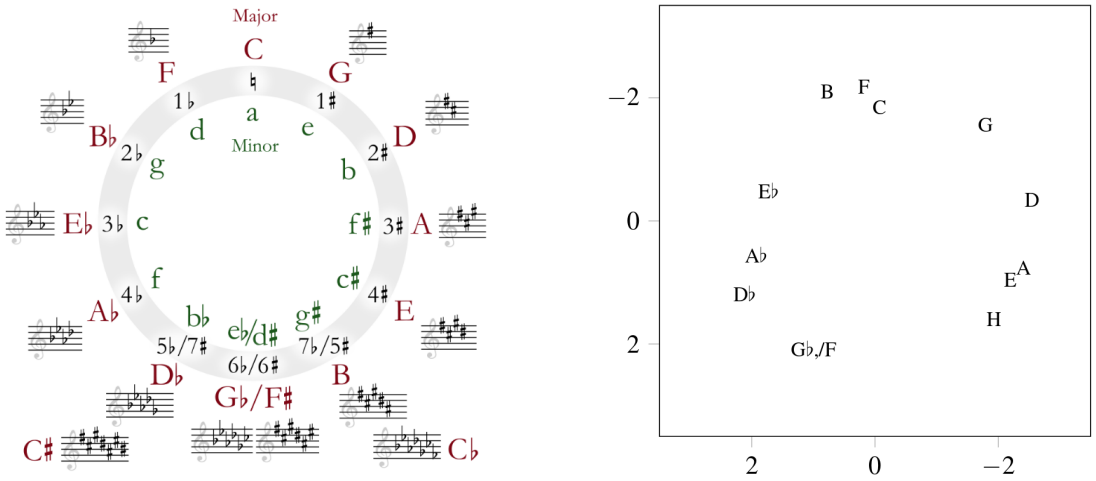
Ycart et al. (2017) studied the influence of different hyper-parameters of a simple LSTM network, such as the number of hidden nodes, the learning rate, the use of data augmentation, on the polyphonic modelling task. Considering full-range piano-roll encodings (88 pitches), the authors also experimented with two common quantisation strategies used to extract them from symbolic music: (i) an *absolute time* quantisation to 10ms; and (ii) a *relative time* quantisation to the 16th note. To ensure a controlled setup, the same LSTM recurrent network with a single hidden layer and sigmoid outputs was considered in all experiments. To sample binary piano-rolls from the model, a simple thresholding mechanism is adopted on top of the sigmoid layer.

The proposed ablative study was performed on two data collections: a synthetic dataset comprising 3-second music clips created by picking notes and chords in the C major scale; and the *Piano-midi.de* dataset, including classical piano music from various composers. For the last dataset, where the duration of pieces varies from 20 seconds to 20 minutes, the authors decided to keep the first minute from each track only. Although this approach is computationally more affordable and permits using larger batches during training, presenting a model with short clips does not allow one to analyse the model's ability to learn long-term dependencies.

The different configurations of the LSTM music model were all evaluated in regard to their predictive capabilities using common classification metrics, i.e. precision, recall and F-measure. From their results, Ycart et al. observed that *absolute time quantization* tends to be more accurate for music prediction, as the short temporal frame (10ms) – a tiny fraction of time for any note or chord, enforces self-transitions (i.e. notes prolonged from previous time steps). The evaluation is thus inflated by the multitude of repeated frames. In this settings, the authors also reported that using a larger number of hidden nodes in the LSTM, as well as augmenting the dataset by transposition to all keys, did not significantly improve the prediction performance of the music model but shortened training times. On the other hand, the music model trained on *relative time* piano-rolls was less performant. Nonetheless, using higher learning rates, data augmentation, and more LSTM cells, not only led to quicker convergence, but also to better results. In the proximity of note changes, this model was also said to spread the probability mass over pitches in the same scale, while ruling out any out-of-scale note.

Another contribution assuming independence between simultaneous notes is JamBot (Brunner et al., 2017). Similarly to (Eck & Lapalme, 2008) for monophonic music modelling, the authors used two LSTM recurrent networks as follows: the *chord LSTM* specialises in modelling chord progressions; a second network – the *polyphonic LSTM*, models polyphonic music while it is conditioned on the predicted chord progression. Hence, the sequence of chords is first predicted and used as a guiding signal for the polyphonic model.

The authors used piano-roll representations quantised to the 8th-note, and assumed that only a chord could be played within each measure. To retrieve chords from each track in the



(a) Illustration of the circle of fifths (in music). (b) Chord embeddings learned by the *Chord LSTM*.

chosen symbolic collection – the Lakh MIDI dataset (Raffel, 2016), an extensive pre-processing procedure was performed. More precisely, data mining techniques were first applied to infer the key and scale from each track. This allowed to filter songs in a major/minor scale, and to transpose them to the same key of C. In doing so, the model only has to learn to predict music in one key instead of twelve – hence no transposition invariance is achieved. Finally, chords could be extracted for each measure by computing the histogram of the 12 notes over that measure, and considering the three most played notes as the resulting chord. For simplicity, only the 50 most frequent chords in the dataset are considered, and the others receives an *unknown* tag.

As mentioned before, the architecture is composed of the following recurrent networks.

1. *Chord LSTM*, processing sequences of chords (chord progression) at the chord level. Each chord is represented as a one-hot vector, which is first passed to an embedding layer, so that the network can learn a meaningful and compact representation of chords from the training data. The embedded chord is then fed to an LSTM, followed by a softmax output layer returning a probability distribution over chords to be played next.

   One of the most interesting results from this model is the embedding space learned from the training data (Brunner et al., 2017). To visualise such embeddings, principal component analysis (PCA) was used to reduce the 10-dimensional chord embeddings into 2-dimensional representations. As illustrated in Figure 4.18b, the visualised embeddings could recreate the circle of fifths, i.e. a diagram that musicians use to visualise the harmonic relationships between chords (Figure 4.18a). Therefore, the model was demonstrated to be able to extract concepts of music theory from songs without any kind of prior musical knowledge. Nevertheless, it is also worth to remark that this property was observed only when musical pieces are transposed to the same key of C major/minor.

2. *Polyphonic LSTM*, implementing a sequence model over piano-rolls with chord conditioning. At each time step, the network receives a binary vector resulting from the concatenation of several features. As any music model using piano-rolls, the first input feature is a one-hot vector encoding the pitches played at the corresponding time step (in this work, an 8th note fragment). To implement the conditioning mechanism, the second and third input features are the embeddings of the chords occurring at the next two time steps, both predicted by the *chord LSTM*. As also done in (Johnson, 2017), a binary vector counting the current relative position within a bar (e.g. from 0 to 7) is also provided to the model. In fact, Brunner et al. argued that this information helps the LSTM to timely be aware of the metrical boundaries, and how many steps remain before the next chord.

   The concatenation of all these input vectors is thus fed to the LSTM, with sigmoid activations in the output layer yielding the probability of each note to be played next. Although note independence among simultaneous pitches is implicitly assumed, the implemented conditioning mechanism – providing relevant harmonic information to the model, helps the *polyphonic LSTM* to disambiguate among notes belonging to certain chords. In addition, the predicted notes are more likely to follow the given chord progression, with the resulting generations potentially achieving more long-term structure capabilities.

The approach proposed in this work is one of the first attempts at learning musical features to better inform the predictions of a music model. Indeed, regardless of the implicit independence assumption, learning chord embeddings and learning to predict chord progressions to guide a polyphonic model can help reducing the number of musically plausible notes to consider for prediction. In other words, it can potentially reduce the model's perplexity at each time step.

4.3 Problem statements

Given the technical limitations and open gaps of the methods addressed in the literature review (Section 4.2), this part outlines and motivates the main research problems in music modelling. Some of these problems were already introduced in Section 4.1, nonetheless the current perspective serves to reinforce the fact that state of art methods fall short in fulfilling these challenges.

The main research problems in the field are organised at two levels: the first groups the open issues encountered during the definition and the evaluation of any music model (Section 4.3.1); the second includes the specific technical challenges making the music modelling problem particularly difficult to address (Section 4.3.2) – explaining the discrepancy between the current methods in the literature and the ideal properties of a music model.

Authors	Architecture	Dataset	Representations	Prior knowledge	Prediction eval.	Generation eval.	Code
Composite architectures: recurrent probability estimators							
Boulanger et al.	RNN-RBM, RNN-NADE	Piano-midi, Nottingham, MuseData, JSB Chorales	piano-roll	-	cross entropy, frame-level acc	discussion	TF
Lyu et al.	LSTM-RTRBM	MuseData, JSB Chorales	piano-roll	-	cross entropy, frame-level acc	discussion	no
Goel et al.	RNN-DBN	Piano-midi, Nottingham, MuseData, JSB Chorales	piano-roll	-	cross entropy	no	no
Vohra et al.	LSTM-DBN	Piano-midi, Nottingham, MuseData, JSB Chorales	piano-roll	-	cross entropy	no	no
Composite architectures: parallel recurrent networks							
Johnson et al.	TP LSTM-NADE, Bi-axial LSTM	Piano-midi, Nottingham, MuseData, JSB Chorales	piano-roll	architecture, additional inputs	cross entropy	discussion	Theano
Alternative representations: simultaneous note in the temporal axis							
Liang et al.	LSTM RNN	JSB Chorales	mod piano-roll	-	token error rate, frame error rate	discrimination	TF
Walder	LSTM RNN	Piano-midi, Nottingham, MuseData, JSB Chorales	midi piano-roll	-	cross entropy	no	no
Alternative representations: distributed representations							
Chuan et al.	Conv AE + LSTM	MuseData	tonnetz piano-roll	representations	no	statistical metrics	TF
Huang et al.	LSTM RNN	MuseData	MIDI and piano-roll	-	no	discussion, discrimination	no
Alternative representations: note onset as a separate feature							
Shin et al.	LSTM RNN	custom pop	text-based	chord support	no	discrimination	yes
Assuming independence among simultaneous notes							
Ycart et al.	LSTM RNN	Piano-midi	piano-roll	-	F-measure, precision, recall	no	TF
Brunner et al.	2 LSTM RNNs	Lakh MIDI	piano-roll	chord support	no	no	Keras

Table 4.2: Summary table of polyphonic music models.

4.3.1 Open issues in music modelling

As introduced in Section 4.1, music modelling involves a number of non-trivial activities and design choices related to: (i) encoding symbolic music as sequences that can be fed to a model (music representations), (ii) designing and configuring a predictive model to process the resulting musical sequences (modelling techniques), and (iii) evaluating the performance of the model so that a comparison with other works in the literature would be possible (*model evaluation*).

Representations In Section 4.1.1 the three fundamental elements needed for the specification of a music representation were detailed, i.e. encoding the evolution of time, note pitch and note duration, and presented the standard approaches for each of these encoding dimensions. In addition, the literature review also provided encoding methods that can emphasise specific musical properties, and hence facilitate the process of learning certain musical regularities. In sum, the technical issues and the methodologies reviewed are summarised as follows.

- *Note pitch representation* are expressed in either *absolute* or *relative* terms. Absolute pitch representations can be implemented as *value encoding*, where a pitch is represented according to its actual features in the audio or music domain (e.g. its frequency value), or as *local encoding*, with the only function to distinguish pitches from one another (e.g. one-hot vectors). Alternatively, a relative representation is commonly based on the concept of *interval change*, hence each note is encoded with the respect to the previous one in terms of the number of half-steps between them (negative or positive).

- *Time and note duration representation* refer to the way music sequences are encoded in order to be processed by a sequence model. Basically, a sequence can be partitioned at the note level, thus considering notes as the most atomic events to process, or using a quantisation method splitting sequences according to a homogeneous temporal unit (either *absolute*, such as 10ms, or *relative*, such as the 16th note). The former approach generally requires using a separate feature to encode the total duration of each note, whereas a quantised representation of time allows to cumulatively express the duration of a note as the number of time steps (or time frames) during which that note is active.

Besides these standard approaches, the contributions from Todd (1989) and Franklin (2006) proposed to enrich absolute representations with concepts borrowed from the fields of musicology and music psychology, thereby providing prior knowledge directly at the representation level. In the same vein, Chuan and Herremans (2018) introduced graphical encodings based on Tonnetz – a visualisation technique used to highlight tonal dependencies in music. As alternative approaches, Lattner et al. (2018) proposed learning interval representations of musical sequences, and A. Huang and Wu (2016) designed a method for learning distributed music representations. To conclude, some representations were intentionally designed to circumventing the complexity of the polyphonic modelling problem (Liang et al., 2017; Walder, 2016).

Therefore, the analysis of the literature indicated that there is still no clear understanding behind the impacts of different encodings of symbolic music on the actual performance of music models. Choosing or designing input/output music representations is indeed the first concern of a modeller, and this has non-trivial implications on the learning process. For this reason, the following research questions should concisely outline the main encoding issues.

Q 1. *How different music representations affect the performance of a music model and which kind of features can be learned from such specific encodings?*

Q 1.a. *Can we explain and measure the effect of enriching music representations with prior knowledge of music? How much prior knowledge is needed to let a model learn adequately?*

Q 1.b. *How does quantisation – the discretization of a track into a number of homogeneous frames, affects the prediction task when varying the granularity of the quantization unit?*

Modelling techniques As detailed in Section 4.1.1, the next step after the choice of music representations is to design a predictive model to process such sequences. Again, the design choices available at this stage are numerous, and a proper understanding of their suitability is currently missing in the literature. Although choosing a particular architecture and an optimal set of hyper-parameters for the model is a general concern for any machine learning task, here we focus on those modelling dimensions and practices that are specific to the research problem.

- *Modelling paradigms and architectures*. Different architectures have been used to implement the autoregressive component of a music model, from Jordan networks and vanilla recurrent neural networks (Mozer, 1994; Todd, 1989), to long-short term memory and self-attention networks (Eck & Lapalme, 2008; C.-Z. A. Huang et al., 2018). With increasing ability to learn long-term dependencies in sequential data, several architectural variations of these networks have also been proposed in machine learning. However, given a dataset of symbolic music with certain properties (genre, degree of polyphony, step-wise movement, pitch range, etc.), a proper understanding of how all these modelling paradigms and techniques perform in such particular contexts is still missing.

- *Supporting prior knowledge*. From the literature, it emerged that prior knowledge of music can be supported in two ways: at the input level – in the form of enriched representations, or by reflecting certain theoretical concepts of music in the model's architecture. This last approach is generally implemented by *providing additional inputs* to the model, either as harmonic conditioning (e.g. providing chords) or metrical conditioning (e.g. providing notes occurring at precise previous time steps) (Eck & Lapalme, 2008; Eck & Schmidhuber, 2002c), or by introducing further connections that are motivated by specific music relationships, as in (Johnson, 2017) for transposition-invariance.

- *Loss function and regularisation*. As we are dealing with a prediction task that is often cast as a classification problem, due to the categorical nature of the most common music representations (e.g. piano-rolls), the standard loss function in music modelling is the *cross entropy*. Although some contributions in the field of music generation extend the loss with music-specific regularisation terms – to enforce certain regularities in the predictions, current methods in music modelling do not seem to adopt a similar approach.

- *Dealing with data scarcity*. A final concern is the scarce availability of large collections of symbolic music – a key requirement for any method based on traditional deep neural networks. As outlined in Section 2.1.3, most datasets are genre-specific and only provide a limited collection of symbolic tracks compared to datasets in other fields, such as computer vision and natural language processing. The most common approach to deal with

this issue is to leverage data augmentation techniques – which comes with the additional advantage of improving model generalisation. For example, a simple method consists in transposing tracks to all possible keys, which in turn facilitates learning transposition-invariant features. Nonetheless, defining augmentation strategies in the context of symbolic music is not trivial, as any perturbation could potentially compromise the musical texture and the theoretical plausibility of a composition. Hence, a critical analysis of data augmentation procedures in music modelling is still left unaddressed.

In light of these considerations, which indicate a unclear view regarding the suitability of the current approaches and the effect of the numerous modelling dimensions available at this stage, the main modelling issues can be formulated as the following open research questions.

Q 2. *How different autoregressive architectures and design choices affect the performance of a predictive model for music and the kind of musical features that can be learned?*

Q 2.a. *How different ways of encoding prior knowledge into the model architecture compare to each other? Can a music model achieve comparable performance and learn similar features without any provision of prior musical knowledge at the architectural level?*

Q 2.b. *Is the minimisation of cross-entropy an exhaustive training objective for a predictive model for music, or should a loss function include music-motivated regularisation terms?*

Besides the former design choices, further considerations should be taken into account if a trained music model needs to be used as a generative system. In fact, training a music model is generally done by gradient descent using the loss computed on the model's output distribution (often at the logits level). It it also common to use *teacher forcing* at training time, feeding the ground-truth tokens back to the sequence model at each time step, regardless of the models' predictions. Nonetheless, taking a (trained) music model to generation requires an efficient method to iteratively sample from the model's learned distribution. In this settings, a music token generated at a certain time step would be used as the model's input at the next time step and so forth. Most works in the literature make use of either *greedy search* – merely selecting the most likely token at each time step; *random sampling with temperature* – where the selection at each step depends on both the probability associated to each music token, and the value of a temperature parameter; or *beam search* with relatively short beam sizes. On the other hand, some polyphonic models use Monte Carlo Markov Chain (MCMC) methods such as Gibbs sampling (Boulanger-Lewandowski et al., 2012; Goel et al., 2014; Lyu et al., 2015) or pseudo-Gibbs sampling (Hadjeres et al., 2016). Therefore, for the sake of completeness, it is also important to retain the following question, which is particularly pertinent to music generation.

Q 3. *How different sampling strategies affect the quality of samples and is the computational cost of a certain procedure affordable for the specific application domain?*

Evaluation. The last open issue in music modelling is the lack of a comprehensive evaluation methodology, intended as a collection of both quantitative and qualitative measures evaluating the predictive capabilities of music models and their generated musical material respectively. The availability of such a framework would make it possible to evaluate the overall performance of a music model in a rigorous, comparable, and affordable way.

As evidenced by the literature review, most current works do not adopt a standard evaluation methodology and often resort to domain-specific strategies or human experiments focusing on the qualitative assessment of generated compositions. Not only this approach involves subjective assessment, which is also costly to obtain, but it also jeopardises the comparison of music models on a large scale. In sum, the current evaluation strategies can be summarised as follows.

- *Quantitative evaluation* concerns the evaluation of the prediction performance of a music model, and is based on the assumption that a model that can effectively predict music – having learnt associations between past and future musical content, can potentially encapsulate notions of music perception and composition. Hence, evaluating the predictive capabilities of a music model provides an indicator of the learned musical features possibly reflecting theoretical properties of music. The most common quantitative evaluation measures in the literature are the *log-likelihood* of the model's predictions on the test set, *frame-level accuracy* (Boulanger-Lewandowski et al., 2012), as well as general *classification measures* such as F-measure, precision, recall and perplexity (Ycart et al., 2017).

- *Qualitative evaluation* is related to the analysis of both the theoretical plausibility and the complexity of model-generated music (a collection of musical sequences sampled from the distribution learned by a music model). In this context, a music model is evaluated by means of techniques borrowed from the field of music generation; this is indeed possible as a music model can also be used for music generation. Overall, this kind of evaluation permits to understand – from the generated musical material, to which extent a model has learned the fundamental rules of music composition and if properties related to musical form, metric and harmony can be manually identified (by human inspection). Given a generated music repertoire, the most common approaches for qualitative assessment are based on *statistical comparisons* with the training data, and *human evaluation* either from domain experts (people with background in music composition) or conventional music listeners via Turing tests (asking whether generated music can be distinguished from human compositions) or blind comparisons (measuring the musicality of pieces generated by different models based on certain criteria).

The main drawback of current quantitative evaluation measures is that they might not be directly comparable whenever two models under evaluation use music encodings operating at

different time resolutions. For instance, if a music model uses note-level representations and another is trained on music quantised at the 16th-note, the resulting average test losses would not be comparable in the first place. In these circumstances, these measures need to be adapted or aggregated in such a way as to enable consistent comparisons at the same level.

As outlined before, qualitative evaluation measures also come with several drawbacks and limitations. For example, a statistical comparison returning a high degree of similarity between generated music and training data might also indicate an overfitting trend or a poorly configured sampling strategy. As for human-based evaluation, a thorough analysis of generated composition carried out by music experts can be time-consuming, hence it generally focuses on a limited set of pieces. Instead, qualitative evaluations using crowdsourcing services (e.g. Amazon Mechanical Truck) usually suffer from filtering participants based on their musical skills, which in turn may compromise the controlled nature of the experiment. Finally, a Turing tests do not allow comparisons between different models, but only one of them in an isolated settings.

In light of these considerations, the presented literature review did not fully address the concerns expressed in Section 4.1.1. This observation implies the following research question.

Q 4. *Is it possible to combine the strengths of quantitative and qualitative evaluation methods with the introduction of novel computational methods that can automatically quantify specific musical properties? Can we design a framework to standardise the evaluation of music models?*

To conclude, all the current issues listed so far demonstrate that there still is a considerable gap in the literature. The modeller is thus left with the following question, which summarises all the specific ones reported so far: "How different properties in music modelling – representations, modelling and learning techniques – contribute to the prediction task and which strategies should be used to timely and objectively evaluate and compare the performance of a model?"

4.3.2 Music modelling challenges

Besides the open issues in the field, which still need a proper treatment to drive novel contributions, the technical challenges of music modelling are currently the main obstacles hindering the progress of research. As detailed in Section 4.1.2, these difficulties are inherently related to the nature of music as a sequential phenomenon, with particular emphasis on its hierarchical organisation and the degree of polyphony and voices defining complex dependencies.

As illustrated in Figure 4.1, technical challenges are organised in a task-based hierarchy, starting with *learning transposition-invariant features* and *long-term musical structure* – the most crucial challenges in music modelling (from *monophonic melody modelling*). At the next level, we find *learning note simultaneities* – the additional technical challenge introduced by *polyphonic melody modelling* due to the need of predicting the occurrence of one or more

notes/pitches (the vertical dimension) at each time step (the horizontal dimension); when considered with the previous level, this can be thought of as learning bi-directional dependencies; Finally, *multi-track music modelling* has to deal with all the issues presented so far in addition to the open challenge of *learning inter-track dependencies*; analogously to the previous case, this generalisation characterises the existence of three-directional dependencies.

Whereas Section 4.1.2 details each of these challenges, here we provide a critical analysis of these issues in relation to the methods presented in the literature review (Section 4.2). By highlighting their limitations, the corresponding open research questions are motivated below.

Learning long-term musical structure

Music entails a rich hierarchical structure at different levels (e.g. note, bar, motif, phrase) and for different musical dimensions (e.g. form, metric, harmony). Regardless of the specific task to address, a music model should ideally be able to relate musical content at different time scales, and retain these associations across the whole length of a piece (without forgetting). Learning musical-structure is not a single objective per se, but entails the following sub-goals.

- *Learning global structure*. Music is conceived and organised at different levels, with musical ideas being coherently introduced, repeated, varied and contrasted across the piece at different temporal scales. Depending on the specific genre and style of a composition, notes and rhythmic cells can be grouped in a *motif*, more motives can define a *theme*, which in turn can contribute to the definition of a *phrase* – a more articulated musical idea possessing a complete musical sense on its own. In the same vein, phrases can in turn originate *melodies*, then *periods*, *sections* and so forth. These segments are thus characterised by hierarchical dependencies and their organisation in a composition generally adheres to the principles of *repetition*, *contrast* and *coherence*. For these reasons, in line with the previous observations on qualitative evaluation strategies, one would expect that samples generated from a music model should ideally exhibit these structural properties: segments of different length should repeat across the piece according to a certain form, whereas the succession of contrasting structures, bringing originality and tension, should be coherent enough to keep the consistency and the main musical idea(s) of the piece.

  As it emerged from the literature review, most contributions attempt to learn musical structure by using gated recurrent units (e.g. LSTM) in recurrent neural networks, or better, by employing self-attention networks as autoregressive methods (e.g. Transformers). Both these approaches are meant to increase the memory capabilities of a music model, thereby retaining a longer temporal context that can help dealing with long and

distant repetitions. In addition, considering that learning global structure is strictly related to learning metrical and harmonic structure, facilitating the latter learning goals can potentially facilitate the identification of repetitions at a larger temporal scale.

- *Learning metrical structure*. Similarly to the way words are carefully organised in a poem (the concept of metre in poetry), music also adheres to metrical structure. Hence, a music model should learn the relationships between different kinds of notes (whole notes, half notes, etc.) and the number of beats in a measure, establishing a precise organisation of musical content in homogeneous frames of relative duration. For instance, most pop songs are built on four-beat meters, where the first and third beats are usually emphasised.

  As done in (Eck & Lapalme, 2008) for monophonic music modelling, in (Johnson, 2017) for polyphonic modelling, and in (Chu et al., 2016; Hadjeres et al., 2016) for multi-track music modelling, specific metrical content can be explicitly provided to the model as conditioning signal (additional input) to facilitate learning metrical structure. In turn, this also makes it easier for the model to predict the boundaries of repeated and contrasted segments, and to localise chord transitions within harmonic progressions.

- *Learning harmonic structure*. Besides the individual composition and the arrangement of chords, the "rules of harmony" also affect the nature of dependencies between successive pitches. In fact, the probability that a model should assign to the next music token should also conform with the previous notes in accordance with the tonality of the piece (defined by the key and scale) at a certain moment (tonality can change across the piece). In other words, tonality imposes the observance of certain intervals, and the extent to which this happens depends on the genre and style of a composition (e.g. dissonances in jazz).

  Providing harmonic support as additional input to a music model, was attempted in (Eck & Schmidhuber, 2002b) for monophonic modelling, and in (Brunner et al., 2017) for the polyphonic case. This is generally achieved by providing the model with a contextual window of chords, at each time step, in order to promote tonal consistency of predictions.

In sum, music structure is hierarchical, complex, highly interrelated with other musical dimensions, and embraces a composition as a whole. Modelling the temporal phenomenon of music thus requires considerable memory capabilities, coupled with the ability to identify and relate structural patterns at all possible time scales. This fundamental requirement poses a considerable challenge for virtually any machine learning method processing a complex and highly structured signal. For music modelling, it represents one of the main obstacle for learning long-term structure, even in the monophonic domain – the "easiest" task in the proposed taxonomy.

From a qualitative perspective, A. Huang and Wu (2016) observed that music generated from recurrent networks lack of global structure: samples tend to be either too repetitive from a

local perspective (e.g. the same short musical idea is repeated indefinitely with little variation) or too contrasting and diverging at different levels. In short, there is little or no distinctive musical idea consistently developing along a generated piece (poor structural coherence).

To the best of our knowledge, and supported by the critical observations found in the literature (Boulanger-Lewandowski et al., 2012; Eck & Schmidhuber, 2002b; A. Huang & Wu, 2016; B. L. Sturm et al., 2016; Todd, 1989), music models based on recurrent neural networks generally succeed at learning local (short-term) musical structure. The advent of self-attention networks in music modelling considerably ameliorated this problem (Vaswani et al., 2017), with Transformer models now capable of generating music at the phrasal level (C.-Z. A. Huang et al., 2018). Hence, one may identify some middle ground and conclude that current music models succeeded in learning mid-term structural dependencies in music. Nonetheless, as learning long-term musical structure still remains a central technical challenge, the corresponding research question is also included as follows.

Q 5. *How can a sequential model learn long-term musical structure?*

As addressing the above question already requires considerable research efforts in light of the progress to date, focusing on this technical challenge should be one of the main priorities in music modelling. Although learning transposition invariant features has yet to be fully addressed, using data augmentation techniques based on transposition to all keys has already proved to be a reasonable expedient to mitigate the problem.

Learning bi-directional dependencies

As detailed in Section 4.1.2, dealing with polyphonic melodies introduce the additional challenge of modelling *note simultaneities*. The joint occurrence of notes follows specific harmonic rules to ensure the consonance of the resulting chords (how a chord sounds when considered as an individual entity) and the adherence to the harmonic structure of a piece (e.g. harmonic progression). Therefore, when considered together, *learning long-term musical structure* and *learning note simultaneities* concur to the definition of a more general challenge, i.e. *learning bi-directional dependencies*. Indeed, the occurrence of a particular note at a particular time affects the probability of other notes occurring at the same time (the vertical dimension), but the prediction also depends on the past musical context (the horizontal or temporal dimension).

From a technical perspective, as a polyphonic melody can be viewed as a sequence of high-dimensional objects, the conditional distribution is said to be multi-modal. As formalised in Equation 4.2, a polyphonic model is trained to learn the joint probability distribution of musical tokens (the concurrent occurrence of e.g. notes/pitches) conditioned on a contextual representation of past musical content. In sum, the literature review (Section 4.2.2) identified the following

categories of methods for dealing with bi-directional dependencies.

- *Composite architectures*, combining two (or more) networks according to a subdivision of tasks: a network learns harmonic dependencies (the vertical dimension), while another specialises in learning temporal dependencies (the horizontal dimension). More precisely, two more granular subdivisions were identified in this group.

  - *Recurrent probability density estimators*, exploiting a recurrent network to temporally adapt the parameters of a probability density estimator – a feed-forward network modelling the vertical dimension, while processing a sequence (Boulanger-Lewandowski et al., 2012; Goel et al., 2014; Lyu et al., 2015; Vohra et al., 2015). The main limitations of these models are the computationally expensive sampling procedure (e.g. contrastive divergence, Gibbs sampling) needed for prediction.
  - *Tied parallel networks* are based on the concurrent use of two recurrent neural networks to process the temporal and the harmonic dimensions respectively, where the topology of connections among them is meant to promote the process of learning transposition-invariant features (Johnson, 2017). However, the resulting architecture is particularly complex, highly specialised, and requires extensive conditioning to support metrical and harmonic structure to the model.

- *Alternative representations* are devised to simplify the polyphonic issue by intervening at the music encodings level according to one of the following methodologies.

  - *Polyphonic elements in the time axis* (Liang et al., 2017; Walder, 2016) is based on unrolling the simultaneous notes building chords from the vertical dimension to the temporal (horizontal) dimension in order to process them as successive events. A limitation of this methodology is the increased length of the resulting sequences, which in turn exacerbates the problem of learning long-term musical structure.
  - *Distributed representations* provide numerical representations of symbolic music intended to replace local encodings, such as one-hot vectors. These representations can either be learned from polyphonic sequences beforehand, then used for music modelling (A. Huang & Wu, 2016), or directly extracted at pre-processing time by leveraging musicologist methods (Chuan & Herremans, 2018).
  - *Note onset as separate feature* is a simple method encoding the temporal occurrence of notes (alias note onset) as a separate feature (Shin et al., 2017). Music is hence regarded as a multimodal sequence where temporal information is explicitly encoded in each token. In doing so, notes with overlapping temporal context would indirectly

belong to a chord. Nonetheless, this approach requires modelling both note onset, pitch and duration – all target features to be predicted by the model at each step.

- *Assuming note independence* identifies the last category of methods relaxing the harmonic dependence among simultaneous notes, thereby treating pitches as statistically independent events (Brunner et al., 2017; Ycart et al., 2017). Hence, a model is left enumerating all possible combinations to find any relevant harmonic patterns.

Overall, state of the art polyphonic models either depend on computationally expensive sampling procedures or use complex models where the amount of prior knowledge encoded directly in the architecture makes it difficult to generalise them to different musical styles and genres. On the other hand, although learning distributed representations can potentially turn useful for polyphonic music, this direction still deserves further investigation. In practice, most current methods intentionally simplify or circumvent the problem of polyphony in such a way that it can be cast to the monophonic case. In most cases, these approaches may set simplistic assumptions on the problem, or elongate the resulting musical sequences – gaining efficiency at the cost of effectiveness. These considerations thus motivate the following research question.

Q 6. *How can a sequential model learn bi-directional dependencies resulting from a sequence of multi-dimensional objects in an effective and efficient manner?*

Learning dependencies between multiple tracks

The most general and challenging task in music modelling is that of multi-track modelling. In this context, each composition is considered as a stream of parallel musical sequences – one for each track/instrument/voice. As no particular constraint is imposed, each track can be either monophonic, polyphonic or even correspond to a rhythmic line. Dealing with the multimodal nature of such signals introduces the challenge of modelling dependencies among tracks, while still considering those in the polyphonic, and for each track. Therefore, all the involved modelling objectives – *learning long-term musical structure* (the temporal dimension), *learning note simultaneities* (the vertical dimension) and *learning inter-track dependencies* (the dimension of instrumentation), contribute to the definition of a broader challenge, that of *learning three-dimensional dependencies* from multimodal sequences.

Learning three-directional dependencies in symbolic music is still a relatively new challenge in the field, which is evidenced by the shortage of multi-track models in the literature (for this reason, a literature review of these methods was not provided, as done instead for monophonic and polyphonic models). Two notable contributions in this domain are DeepBach (Hadjeres et al., 2016) and SongFromPI (Chu et al., 2016). Both these methods use a neural network

architecture to model each single track, but it is also assumed that all tracks are monophonic. This assumption clearly removes one of the three learning objectives of the multi-track problem, which thus remain an open research question.

Q 7. *How can a sequential model learn three-directional dependencies resulting from a stream of parallel tracks made of multi-dimensional objects?*

Modelling multi-modal and highly-dimensional signals is one the major quests in audio processing, and is also particularly relevant to other fields, such as computer vision. For instance, the multi-track music modelling problem can be compared to having several video streams recorded from different cameras but related to the same phenomenon. These technical challenges are thus common to a multitude of tasks in machine learning, making music modelling an ideal application domain where the complexity of the music signal can generalise other problems and foster the advancement of research in manifold directions.

4.4 Music modelling research questions in this thesis

This chapter presented an in-depth analysis of the main open issues and technical challenges in music modelling, motivating their nature in light of the limitations currently found in the literature. Nevertheless, it is worth remarking that the thesis does not intend to address all the open research questions in the field. Some of them are directly addressed, whereas others are potentially ameliorated thanks to the introduction of new methods and tools that can support the identification of technical requirements and the evaluation of the proposed solutions. In particular, as contextualised in the introduction (Section [1.4]), this research project focused on the open issues of improving the understating of modelling techniques (Q 2), and most importantly, on the design of computational methods for the automatic evaluation (Q 4) of music structure complexity – a key requirement for the assessment and the comparison of methods addressing Q 5. In addition, we also investigated how the detection and disentanglement of structural properties can affect to other tasks in music information retrieval.

Part III

Contributions

Chapter 5

Structure in music modelling: an upstream approach

As motivated in Section 2.1, a music model is a computational tool allowing a large number of applications ranging from style modelling and score transcription to automatic music composition. Regardless of the specific application domain, the first step after training is to evaluate the predictive capabilities of the music model and analyse the type of features it managed to learn. Ideally, this should be done using different datasets and encodings of symbolic music, as models can be particularly sensitive to the way music is represented, and to the distinctive properties of the music collections under study. The latter can be related to general attributes of the collection (e.g. number, quality and completeness of the transcriptions) or intrinsic to the music it contains (genre, style, duration, degree of polyphony, structural complexity, etc.).

If done according to a standardised methodology, this kind of evaluation would enable an in-depth analysis of music models, and a systematic comparison to the state of art (c.f. Section 4.1.1). Nonetheless, the literature review demonstrated how evaluation still represents an open issue in the field, and emphasised the lack of such an evaluation framework (Section 4.3.1).

Based on these premises, the first contribution aims at establishing a standardised methodology for the systematic evaluation of music models, before bringing them to any application domain (e.g. music generation). This also provides the computational infrastructure to address RQ 1. Therefore, this chapter introduces the main components of our evaluation framework, starting from the architectural types and the symbolic music encodings currently supported. Then, we conduct a large scale experiment to demonstrate how the framework can be used to gain useful insights from the modelling perspective, and for benchmarking novel architectures. The contributions in this chapter extend the work presented in (de Berardinis, Barrett, Cangelosi, & Coutinho, 2020), which introduced one of the architectures in the framework.

5.1 MOlliE: The music modelling framework

As detailed in Section 4.1, the task of music modelling revolves around three fundamental activities, which are carried out in the following order: choosing (or designing) a machine-readable format to *encode symbolic music* and performing domain-specific preprocessing on the resulting sequences (including data augmentation); designing a suitable architecture for *modelling* the so-encoded musical sequences and setting up an effective training procedure (loss, regularisation terms, optimiser, learning rate schedules, etc.); and finally, *evaluating* the music model on a test set to measure its generalisation capabilities and analyse the kind of features it learned to see if any music-related patterns were retained. These steps are illustrated in Figure 5.1, and put in relation with music generation to emphasise the dual role of evaluation.

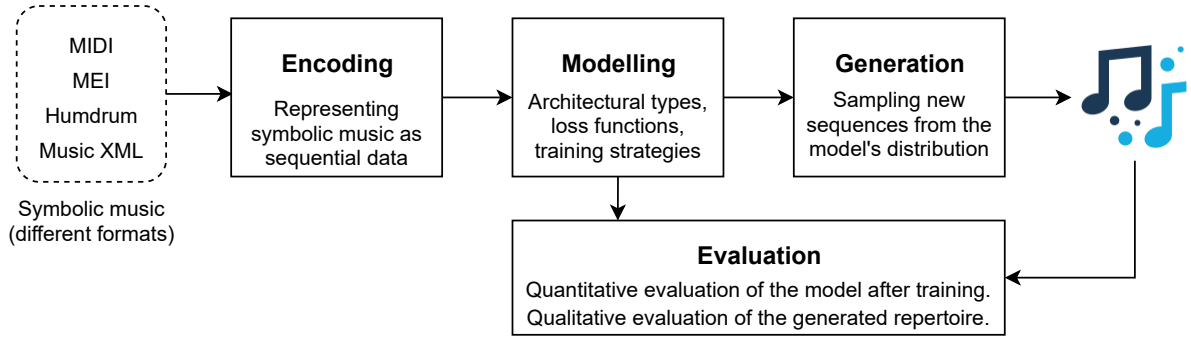


Figure 5.1: The main steps in the music modelling pipeline, also in relation to music generation.

To the best of our knowledge, MOlliE is the first music modelling framework providing a comprehensive set of utilities and core functionalities to facilitate all the aforementioned activities. These include a collection of symbolic music encodings, state-of-art neural network architectures, and visualisation utilities allowing experimenters to design, evaluate and compare their music models in a controlled environment. To simplify the expected workflow, the framework is organised in three main modules – consistently aligned with Figure 5.1, and providing a number of building blocks to complete the corresponding steps.

Encoding. To gain preliminary insights into the data collections under analysis, the module includes functions to extract descriptive statistics on the respective musical material, such as the distribution of pitches, intervals, duration, time signatures, tempo changes and note density (alias degree of polyphony). Not only these are useful to inform the subsequent steps (choice/design of encodings and architectures), but they can also support the interpretation of the modelling performances at evaluation time. Besides dataset utilities, MOlliE provides a predefined set of encodings – functions transforming symbolic music, originally represented in different formats, into sequences that can be fed to a music model. Each encoding is based on a vocabulary to represent a certain music event at the most granular level

(usually a temporal or metrical fraction of a note/chord), hence it produces distinctive sequences compared to other encodings (c.f. Section 4.2.1). Finally, the module also includes preprocessing utilities for filtering infrequent or undesired musical events and adapting sequences to match desired properties, together with data augmentation operations.

Modelling. To facilitate the design of novel architectures and enable their comparison with current state of the art music models, MOlliE provides a collection of neural architectures that are representative of several works in the literature (c.f. Section 4.2). Given the predominance of these computational methods, the framework focuses on a specific family of architectures – autoregressive (AR) models for music (alias predictive models or sequence models for music). As detailed in Section 4.1.1 these models are trained to predict the next music event in a sequence given a contextual representation of the previous events. The main architectural types supported in MOlliE are based on variations of recurrent neural networks and self-attentions networks, including hybrid architectures that further generalise the former. The core modelling paradigms on which these architectures are defined can also be reused by the experimenter as building blocks of their new music models. Hence, this modular approach promotes a more controlled testbed for the comparison of music models.

Evaluation. This last module provides a set of tools for the evaluation of music models after training, covering both quantitative and qualitative evaluation methods. The former are meant to measure the predictive capabilities of music models, and are common in any machine learning task (c.f. Section 4.1.1). As this information is usually given by the test loss of the models' predictions at the token level[1], the module provides a set of functionals for analysing key properties of the loss sequences, such as variability, predictability and stationarity. Concerning the qualitative methods, the module includes visualisation tools for the inspection of models on test sequences. These were specifically designed to allow experimenters to gain more insights on the structural features learned by the model.

In sum, thanks to our framework, different modelling paradigms, representations of symbolic music, and training procedures can be tested on a number of datasets, allowing the evaluation and the comparison of novel AR models for music with the SoA. Therefore, MOlliE has dual purpose: (i) it can support general explorative studies, aiming to inspect the impact of the numerous modelling dimensions and design choices on the predictive capabilities of music models; and (ii) it provides an evaluation tested offering several functionalities and allowing experimenters to benchmark their methods with SoA architectures in a fast, controllable and reusable way. In relation to Figure 5.1, it is also worth to remark that MOlliE focuses solely on

[1] In a sequence-to-sequence model, the loss is computed for each token (music event), hence the loss of a given musical sequence (a composition, a transcription of a performance, or a part of them) is also a sequence.

music modelling, hence it does not address the generation domain. For the latter, several sampling techniques and further evaluation criteria would be needed, Nevertheless, the automatic evaluation of music generation methods, based on the structural properties of their compositions, is also of interest and will be discussed in the next chapters (c.f. Chapters 6 and 7).

5.1.1 Supported music encodings

Despite the fact that symbolic music can be represented/provided in different formats, such as MIDI, MusicXML, digitised scores, ABC, and so forth (Section 2.1.3), a formal representation of their content – a mathematical object (a tensor), is needed to present such data to a music model. Although the terminology may be inconsistent in other fields, these objects are known as music encodings and each of them results from a specific encoding function – providing a unique mapping from arbitrary symbolic representations.

Devising new music encodings is not a trivial task, as it involves different design choices which could emphasise certain musical properties (intentionally or not). As detailed in Section 4.2.1, these include the representation of note pitches, their duration, and the granularity of the music tokens to process (the most atomic events in the resulting sequences). In addition, input sequences can be designed in a way to contain additional music material that does not appear in the corresponding output sequences (those that the model has to predict). This technique is called *conditioning* and is often used to facilitate the modelling task, as well as to achieve more control over the model at inference time. Therefore, encoding symbolic music is not as easy as encoding words in NLP, hence this complex task necessitates proper treatment.

In MOlliE, the user is free to define their own music encoding, starting from a set of available constructs and extending some base classes (software interfaces) that ensure the interoperability of different representations. Alternatively, they can simply reuse one of the provided (built-in) music encodings. At the current time, the framework supports two main encoding types that can handle single-track polyphonic music and allow for additional control via parameterisation (e.g. the unit of quantisation to adjust the granularity of the resulting music tokens). Both the currently supported music encodings are MIDI-based, meaning that they represent music as a stream of time-stamped events. Hence, chords are not represented as individual events, but rather as special sub-sequences where their their note constituents are unrolled horizontally. Although this approach elongates the encoded sequences, it makes it easier to model polyphony, which should otherwise be learned as the joint probability distribution of all pitches being played (or released) at the same time step – the approach of piano-roll representations.

Polyphony encodings

This first class of MIDI-based encodings is representative of several works in the literature, such as those used in `BachBot` (Liang et al., 2017), With the polyphony encodings (alias *step-based encodings*), each piece of music is first quantised according to a metrical unit (metric-based quantisation) and represented as a sequence of MIDI events delimited by `START` and `END` symbols. If music sequences are split in shorter fragments, this allows the model to recognise if a sequence fragment opens a new composition, when it is approaching the end of a piece, or simply, if intermediate musical material should be modelled.

As mentioned before, chords are represented as special sub-sequences of note events enclosed within `STEP_END` symbols. These special delimiters are needed to indicate that all notes belonging to a step are associated to the same time step. In other words, time is assumed to advance (for the metrical unit chosen as quantisation level) after each step. Within a step, notes are sorted by pitch in descending order, as if the chord is arpeggiated. For example, using the 16th note as unit of quantisation (4 steps per quarter note), a sequence containing only a C major chord with a duration of a minim (half note) would be encoded as follows.

```
START, (NEW_NOTE, 67), (NOTE_NOTE, 64), (NOTE_NOTE, 60), STEP_END,
   (CONTINUED_NOTE, 67), (CONTINUED_NOTE, 64), (CONTINUED_NOTE, 60), END
```

Technically, to distinguish between new and continued notes, each pitch number is associated to two different events. Assuming the full MIDI pitch range (128 pitch events), the size of the vector encoding a musical token – corresponding to the cardinality of the encoding vocabulary, is computed as: 128 (new notes) + 128 (continued notes) + 2 (sequence delimiters) + 1 (step delimiter), yielding 259 events in total.

Performance encodings

The performance encodings were introduced by Google Magenta and used in state of the art music models, including the Music Transformer (C.-Z. A. Huang et al., 2018). These encodings most closely match the MIDI standard, as they replace the note continuation events of the polyphony encodings with `note-off` events. Note pitches are thus assumed to remain active from the moment they are played until they are explicitly released.

Similarly to the polyphony encodings, musical sequences are still quantised into fixed-size temporal units. However, two key differences are peculiar to these encodings: (i) both metric-based (relative) and time-based (absolute) quantisation are supported, hence they can either be chosen depending on the desired properties of the musical features that will be learned by models (tempo invariance vs modelling tempo variation in performed music); (ii) special time-shift

events are introduced to allow the model to skip forward in time to the next note event. Thanks to the flexible representation of time advancement, the encoding "consumes" more events in sections with high note density, and fewer events when notes are held during slow passages.

Overall, these encodings promote a more parsimonious usage of musical tokens, as they tend to produce shorter sequences compared to the polyphony counterparts. This also frees the model from having to learn to repeat redundant/unchanged steps, and facilitate the use of finer quantisation levels. Together with the possibility of representing note velocities, these last properties motivate the suitability of these encodings to performed music.

Concerning the cardinality of the vocabulary, the minimum number of events to cover the full MIDI pitch range is still 128 (note-on) + 128 (note-off), if velocity events are not taken into account. In addition, assuming that metric-based quantisation to the 16th note are used, skipping up to a whole note would require 16 time-shift events (272 events in total). Alternatively, using absolute quantisation with resolution of 25ms would require 25 time-shift events to skip up to 1s of music (281 events in total).

5.1.2 Supported modelling paradigms

The main architectural types of music models supported in MOlliE – outlined in Table 5.1, are based on three fundamental modelling paradigms in machine learning, i.e. *recurrence*, the presence of an *inductive bias*, and *attention*. From a general stance, a modelling paradigm can be defined as a mechanism allowing the model to relate information across different sources (intermediate features) in a particular way, hence introducing significant changes in the architecture.

Architectural type	Recurrence	Inductive Bias	Attention
Basic LSTM	yes	through time	-
Attention LSTM	yes	through time	past hidden states
Linear Transformer (T)	-	-	self-attention
Linear Universal Transformer (UT)	-	through depth	self-attention
Long-Short Term Universal Transformer (LSTUT)	yes	through time, through depth	self-attention

Table 5.1: Overview of the architectural types of music models supported in MOlliE, emphasising their relation with the main modelling paradigms for sequence modelling.

As outlined in Section 3.1, recurrence is one such paradigm which is extensively used in sequence modelling. The main idea consists in introducing recurrent connections in the architecture that allows the model to maintain a contextual memory representation of the previous tokens. An inductive bias, instead, is a tendency to learn iterative or recursive transformations.

For example, adaptive computation time (Section 3.5.2) can be thought as an architectural component allowing the network to pass through the same layers multiple times (the number of passes varies depending on the context). Nevertheless, inductive bias is a property that recurrent networks posses by definition. Indeed, a RNN learns a function of both the input and the hidden state, which is then applied iteratively across the input sequence (recurrence through time). Finally, attention (Section 3.5.2) can be described as an interpretable mechanism to learn similarity functions that can be used to fetch contextual information from different inputs or parts of the network. As attention is a fairly general concept, further modelling paradigms are actually defined from it, such as *attentional interfaces* and *self-attention*.

Based on the architectural types outlined in Table 5.1, we can distinguish between three categories of music models: *long-short term memory networks* (recurrent neural networks using LSTM cells), *self-attention networks* (attention-based feed-forward networks), and *hybrid architectures*. Among these models, the LSTM and Transformer networks are a staple in machine learning research and have been extensively used in music modelling. Conversely, the universal Transformer (UT) has never been applied to music modelling, and the long-short term universal Transformer (LSTUT) is our novel hybrid architecture.

The following subsections present each category together with the specific instances of music models that are part of our reference benchmarking testbed.

Long-short term memory networks

As detailed in the literature review (Section 4.2), the LSTM is at the core of most music models, and it was the predominant architecture before the advent of Transformers. Although the backpropagation-through-time (BTT) algorithm used for training recurrent networks is computationally demanding (c.f. Section 3.3), these models are particularly efficient at inference time – making them appealing for automatic music composition and real-time accompaniment.

Model name	LSTM layer size	LSTM layers	Attention layer	No. of parameters
lstm-2	256	2	no	1.191.184
lstm-3	256	3	no	1.717.008
lstm-4	256	4	no	2.242.832
lstm-att	256	2	yes	1.256.976

Table 5.2: Overview of the supported LSTM recurrent neural networks.

From Table 5.1, we can see that MOlliE supports two LSTM architectures: the traditional implementation from Hochreiter and Schmidhuber (1997) (Basic LSTM), and another using a simple attention mechanism (Attention LSTM). The former is representative of several works

in the literature, from the seminal work of Eck and Schmidhuber (2002a) modelling harmonic and melodic sequences of jazz music, to modern architectures for polyphonic music (Johnson, 2017). For the Basic LSTM, the framework includes three music models with different configurations (`lstm- 2`, `3`, and `4`), allowing the experimenter to analyse the effect of stacking an increasing number of recurrent layers (see Table 5.2). Instead, the Attention LSTM extends the original model with an attentional interface right on top of the recurrent layer, thus allowing the network to contextually fetch information from past hidden states (c.f. Section 3.5.2). As advocated in (Elliot, 2016), this attention mechanism is supposed to improve the long-term modelling capabilities of the architecture, as the network can (learn to) utilise past information rather than exclusively relying on the LSTM's cell state.

Self-attention networks

Since the advent of the Transformer (Vaswani et al., 2017), self-attention networks (SANs) have dominated research in sequence processing thanks to their increased ability to retain long-term dependencies. As this is related to learning long-term structure in music, they also became widely used in music modelling and automatic composition (C.-Z. A. Huang et al., 2018). More precisely, a SAN for sequence modelling typically refines and extends the encoder part of a Transformer. This is why a full encoder-decoder architecture is more suited to sequence transduction tasks – where input and output sequences are defined over different vocabularies (e.g. machine translation).

Our framework supports two SAN architectures: a computationally improved version of a Transformer, and another based on the Universal Transformer (UT) from (Dehghani, Gouws, Vinyals, Uszkoreit, & Łukasz Kaiser, 2018). Whereas the first architecture has become SOTA in music modelling and is similar to the Music Transformer (C.-Z. A. Huang et al., 2018), the latter provides a generalisation of the former and, to our best, is novel in the music domain.

Model name	SA layers	Encoding steps	Embedding dim.	Att. heads	Att. hidden size	No. of parameters
linear-transformer	3	1	128	4	512	663.952
linear-universal-transformer	1	4	128	8	512	267.920
linear-universal-transformer-tall	1	12	128	8	512	267.920
linear-universal-transformer-wide	1	6	128	16	512	267.920

Table 5.3: SAN models supported in our framework (all using linear self-attention).

The UT generalises a Transformer to address one of its limitations – the lack of an *inductive bias*. This is achieved by adapting the architecture in such a way as to allow the UT to recursively iterate through the encoder. More technically, as can be seen in Figure 5.2, the transformer layer is repeated as a loop according to the number of *recurrent steps* (alias *encoding steps*). This originates a stack of encoder (E) layers sharing the same parameters. Trivially,

when the number of recurrent steps is 1, the UT reduces to a vanilla Transformer encoder. Not only does this property promote parameter sharing, but is also appealing for music modelling, due to the hierarchical nature of tonal music which makes it suited to such transformations.

The transformer architecture originally proposed in (Vaswani et al., 2017) has also been criticised for being computationally inefficient The computational complexity of the vanilla transformer is proportional to the square of the sequence length, which becomes prohibitive for considerably long sequences. To address this issue, both our architectures implement the attention mechanism from (Katharopoulos, Vyas, Pappas, & Fleuret, 2020), which makes the complexity linear in the sequence length. This is achieved by defining the attention among two tokens as the dot product between the queries and keys, which permits a factorisation. In doing so, the keys are summed across all sequence positions, then multiplied by each query vector separately, followed by a suitable normalisation.

Hybrid architectures

In the original Transformer, encoding the temporal relationships of sequence elements, prior to the self-attention sub-layer, is done by manually injecting this information for each token – a procedure known as *positional encoding*. The reliance on positional encodings has two consequences: the model struggles with both learning a precise representation of the input, and learning to reconstruct a precise representation of the output. In addition, transformers rely on having diversity of attention heads, which is needed for learning across different time scales, although not guaranteed as shown in (Li, Tu, Yang, Lyu, & Zhang, 2018).

Model name	SA layers	Encoding steps	Embedding dim.	Att. heads	Att. hidden size	No. of parameters
lstut	1	4	128	8	512	531.600
lstut-tall	1	12	128	8	512	531.600
lstut-wide	1	6	128	16	512	531.600
double-lstut	2	4	128	8	512	861.456

Table 5.4: Overview of the novel long-short term UT (LSTUT) models supported in MOlliE.

To address these issues, a novel architecture was introduced – the long-short term universal transformer (LSTUT). As illustrated in Figure 5.2, compared to a vanilla Transformer the LSTUT adds recurrence in two ways: (i) leveraging the UT's weight sharing mechanism to perform multiple encoding steps at a linear cost (*recurrence in depth*); and (ii) integrating recurrent neural networks upstream and downstream the transformer layer (*recurrence in time*).

In point of fact, recurrence in time is important to learn local structures from sequences – a learning objective for which RNN typically excel. In the LSTUT, this is achieved with the inclusion of two LSTMs between the linear UT layer. The LSTM on the lower level has two main

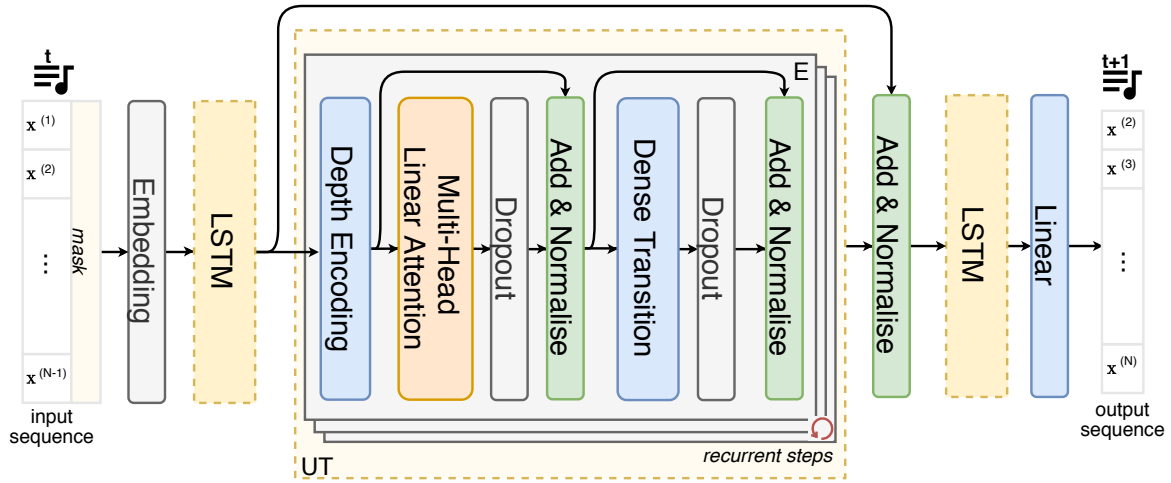


Figure 5.2: Illustration of the LSTUT architecture for music modelling. For "dense transition" we mean a stack of fully connected layers which act as a transition function. The circled addition signs represent residual connections.

benefits: (i) it temporally contextualises the input data, thus replacing the positional encoding; (ii) it facilitates the process of modelling local structures by learning a higher level representation of the input sequence, e.g. labelling chords. The UT then distributes this information across the entire input in a global fashion. The LSTM on the top level is used to "re-localise" the output of the UT, thereby reconstructing it to a high resolution, e.g. outputting chords.

As outlined in Table 5.4, the framework supports four LSTUT models – all sharing the same embedding dimension and attention hidden size in the encoding layer(s). Nonetheless, the first three models differ in the number of recurrent steps (`lstut-tall`) and attention heads (`lstut-wide`). Finally, the last model – the `double-lstut`, consists in a stack of two `lstut`.

5.1.3 Technical details and use of the framework

MOlliE is designed to encourage the reuse of all its modules, from the data pre-processing pipelines and the encoding utilities, to the architectural components of the music models and their evaluation after training. To facilitate large scale experiments, it provides utilities for parallel processing and scheduling of GPU resources. The codebase is released as an open-source project and its code is available at `https://github.com/jonnybluesman/mollie-tf`.

The framework is written in Python 3.7, and the supported models are implemented using Tensorflow 2.2 (Abadi et al., 2015). For data processing, we used Magenta[2], which is part of the Tensorflow's ecosystem, and the `noteseq` library. The evaluation utilities enabling the qualitative analysis of music models make use of `librosa` 0.8 (McFee, Lostanlen, Metsai, et

[2] https://magenta.tensorflow.org/

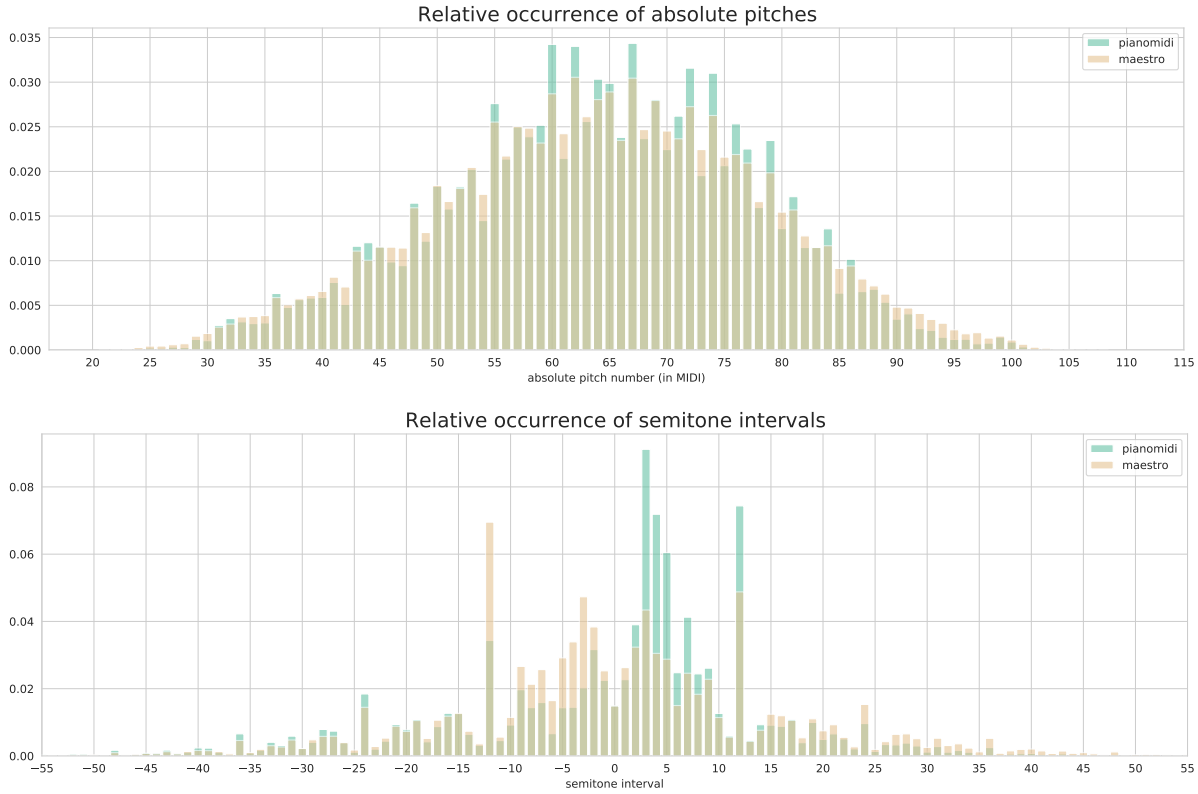


Figure 5.3: Global dataset statistics: relative occurrence of absolute pitches (expressed in MIDI number) and semitone intervals between successive notes – a proxy to step-wise motion.

al., 2020) and `pretty_midi` (Raffel & Ellis, 2014) for visualisation purposes.

5.2 Experimental evaluation

To demonstrate the potential use of the framework, this section reports an experiment benchmarking the main music modelling architectures supported in MOlliE on a large dataset of symbolic music. In particular, the evaluation is specifically focused on the models' abilities to learn long-term structural properties of music. Therefore, this experiment allows to analyse how different modelling paradigms, and neural network architectures influence the music modelling task in relation to the problem of learning structural properties. In this regard, the experimental results also provide the rationale to address RQ 1.

In line with previous works (Elliot, 2016; C.-Z. A. Huang et al., 2018), we hypothesise that *recurrence* and *attention* are both key components for learning short- and long- term dependencies from music. Our hypothesis can be decomposed as follows: (i) *attention* is key for learning long-term structure; (ii) *recurrence* facilitates the discovery of short-term structure; (iii) *inductive biases* improve the ability to learn long-term structure.

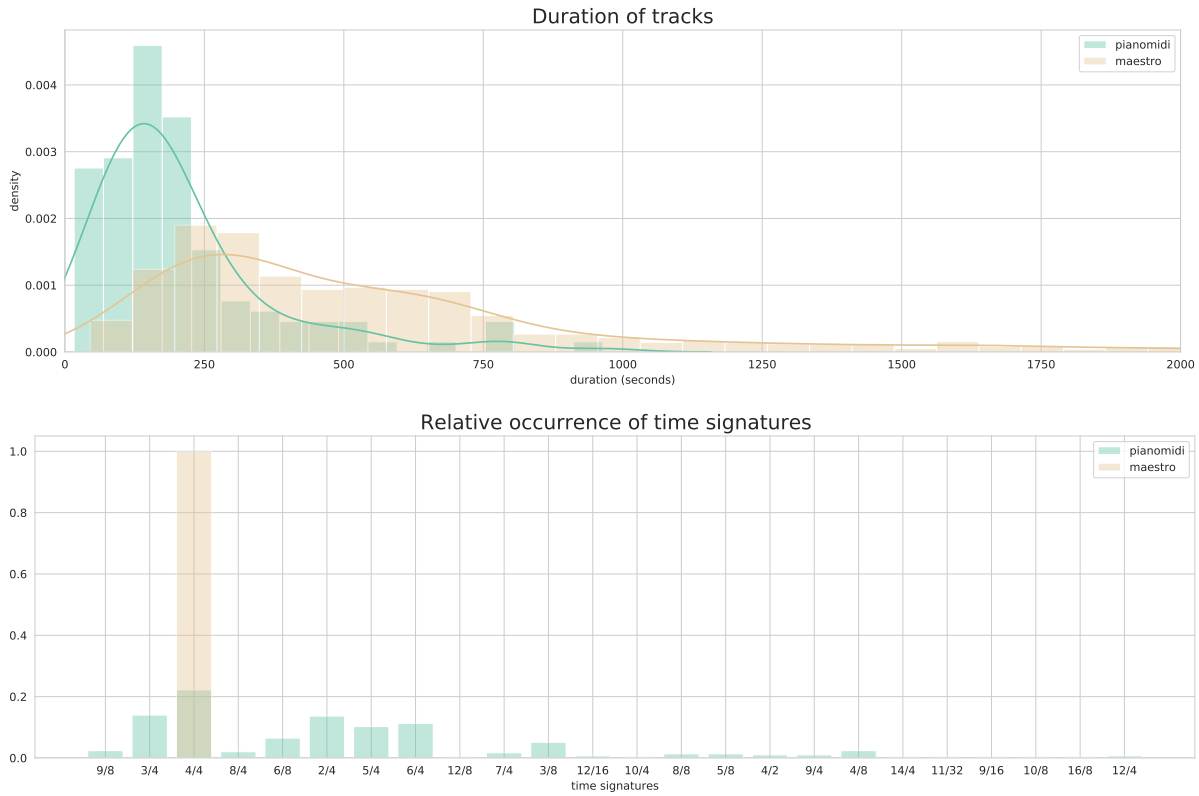


Figure 5.4: Distribution of track durations (*top*) and time signatures (*bottom*) per dataset.

To test this hypothesis, the experiments are organised as an ablation study, where the different architectures under evaluation implement one or more modelling paradigms as reported in Table 5.1. In particular, the Basic LSTM provides *recurrence* and *inductive bias through time*, whereas the Attention LSTM adds an *attention* mechanism (Bahdanau et al., 2014) allowing the LSTM network to contextually access its past hidden states. Both these two models were already studied in music generation experiments by Elliot (2016), where the latter achieved structurally more realistic generations according to the authors. Nevertheless, these architectures only differ on the attention mechanism. Coming back to our quest, the next architectures under evaluation are the Transformer, providing *self-attention*, and the Universal Transformer, adding *inductive bias through depth* (c.f. Section 5.1.2). Conversely to the LSTM networks, both these architectures lack recurrence. Finally, all these modelling paradigms are present in the LSTUT; hence we expect this architecture to achieve the best music modelling results.

5.2.1 Dataset

All the music models in our experiments were trained and tested on the *Maestro Piano-e-competition* dataset (Hawthorne et al., 2019), which includes classical piano performances in

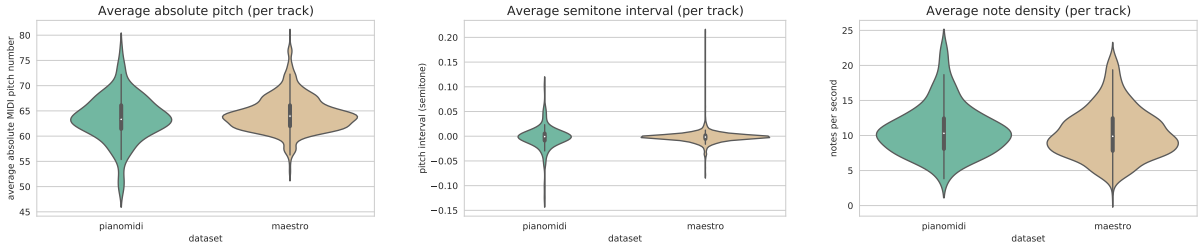


Figure 5.5: Per-track dataset statistics: distribution of the average absolute (*left*) and relative (*centre*) pitch in each track; and average number of notes per second, alias note density (*right*).

symbolic format (MIDI). Considering the nature of the musical content (transcribed performances), as well as some key musical properties (e.g. duration of tracks, pitch distribution), Maestro provides a challenging testbed for music models. This observation is more evident when the dataset is compared to other collections, such as *Pianomidi* (Bernd, 1996); a first comparison of the basic properties of these datasets is provided in Table 5.5.

Pianomidi is a relatively small collection compared to Maestro, yet is considered a standard testbed and has been extensively used in polyphonic music modelling over the past 20 years. On the other hand, Maestro provides a larger selection of piano performances (1276 pieces) which is still smaller compared to recent datasets, such as GiantMIDI (Kong, Li, Chen, & Wang, 2020); although the latter counts 10853 tracks, these symbolic pieces are the result of an automatic transcription process using SOTA computational methods. Therefore, for our experiments, we chose Maestro as the representative of a large collection, because the music has been directly sampled during real performances and should contain more reliable musical content – especially that pertaining to structural changes (e.g. elongated notes).

Dataset	Training tracks	Validation tracks	Test tracks	Total tracks	Average duration
Pianomidi	87 (5:16:39)	12 (0:35:00)	25 (1:19:25)	124 (7:11:42)	208.88 $\pm$ 168.51
Maestro	962 (6 days, 15:14:32)	137 (19:25:46)	177 (19:58:46)	1276 (8 days, 6:39:04)	560.45 $\pm$ 442.88

Table 5.5: Number and duration of symbolic tracks in the Pianomidi and Maestro datasets.

In sum, the dataset provides an ideal test-bed for our experiments due to (i) the structural complexity of the compositions; (ii) the inclusion of long music pieces; (iii) the fact that part of Western classical understanding of musical form lies in performative interpretation of a score.

Analysis of the musical sequences

Using the data analysis tools of MOlliE, a set of statistics were extracted to gain further insights on the musical properties of the dataset under study. Although the experimental setup is focused on Maestro, the same statistics are also extracted for Pianomidi for the sake of comparison. As

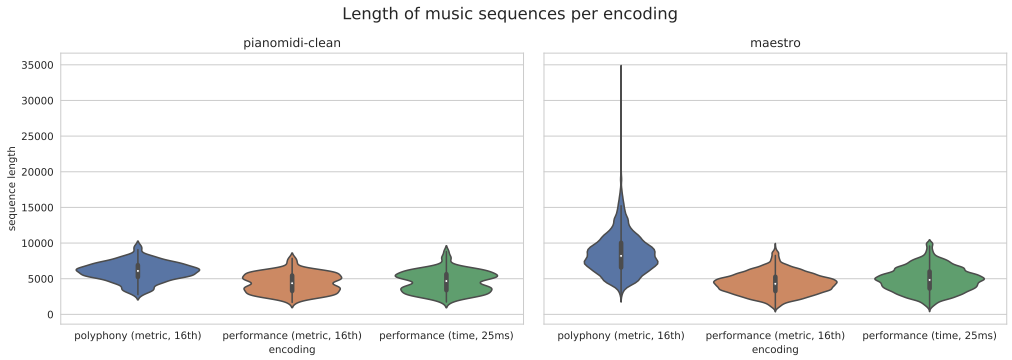


Figure 5.6: Length of the encoded sequences for each dataset and for each encoding type.

part of the music modelling workflow, this analysis is generally helpful to inform the design/-choice of encodings, and to a lesser extent, the architecture of music models. In addition, it can provide rationale to support the interpretation of the music modelling results, and the analysis of the models' behaviour or its learned features.

The first set of statistics is illustrated in Figure 5.3 and offers a collection-level overview of the global pitch space. More precisely, these histograms are computed on each collection as if all its pieces were a single one (no aggregation at the track-level is considered). The first histogram gives the relative occurrence of absolute pitches, where each bin is associated to a distinct MIDI number. Analogously, the second histogram provides a temporal perspective of the pitch space, by representing pitches as semitone intervals between successive notes (a way to visualise the global step-wise motion). From these plots, we notice that both the Pianomidi and Maestro collections have similar absolute pitch spaces, despite the limited size of the former. Nonetheless, when looking at the semitone intervals, the step-wise motion of Maestro has a more descending trend; whereas Pianomidi exhibits the opposite tendency.

Another set of statistics focuses on two central properties: the duration of tracks, and the time signatures that are explicitly encoded in the symbolic pieces (rather than being inferred). As can be seen from Figure 5.4 and Table 5.5, the distribution associated to Maestro indicates that the collection has much longer tracks with more diversified durations compared to Pianomidi. On the other hand, the latter has a more complex metrical texture in light of the diverse time signatures that can be found; whereas all Maestro's music is in 4/4.

To conclude, Figure 5.5 offers a per-track overview resulting from the aggregation of some statistics. The first two plots confirm that tracks from both datasets have similar average pitch and semitone interval. Interestingly, the narrow distributions associated to Maestro indicate that its music tends to gravitate around the current pitch – meaning that tracks have shorter step-wise motion or a higher degree of consecutively repeated pitches. In contrast, Pianomidi exhibits more diversity in the use of pitches and larger step-wise motion, on average.

From symbolic music to encodings

To be ingested by a music model, symbolic music should be first converted to mathematical representations using a specific encoding function. During this exploratory stage, we compute and analyse all the encodings supported in MOlliE: the polyphony and the performance encodings. For the latter, both the metric- and time-quantised versions are considered (c.f. Section 5.1.1).

Prior to being processed by these encoding functions, each track is first quantised to the 16th note for both the two metric-based encodings, and to 25-second bins for the time-based performance encoding. To be computationally manageable, the quantised tracks are split into non-overlapping sub-sequences of 180 seconds – a duration we believe sufficient for testing the models' structural abilities from short- to long-term scales. Finally, as a standard data augmentation strategy, sequences in the training and validation sets are transposed to all keys.

The distribution of sequence lengths resulting from the each encoding is illustrated in Figure 5.6 for both datasets. For Pianomidi, the polyphony encoding produces sequences with average length of 5977 ± 1337, whereas the performance encodings yield 4377 ± 1341 and 4577 ± 1468 for the metric and time versions. For Maestro, the average sequence length of the polyphony encoding surges to 8543 ± 2903, whereas it remains stable for the performance encodings, accounting to 4324 ± 1468 for the metric-quantised, and 4872 ± 1704 for the time-quantised version. Therefore, despite the significantly longer tracks in Maestro, the performance encodings produces sequences of comparable length. As Maestro was found to contain music with lower note-density (Figure 5.5), the performance encodings can keep sequences compact by leveraging the time shifts events to efficiently handle elongated notes and silences.

To conclude, considering the main goal of these experiments, we focus on the worst-case scenario, when music is represented via the most verbose encoding – the polyphony encoding.

5.2.2 Evaluation measures

All models were evaluated on the music modelling task, quantifying their ability to predict symbolic music. This was done by computing the cross entropy loss and perplexity of each model on the test set. Cross-entropy is the dominant measure in machine learning for training and evaluating classification models, and it is also extensively used to evaluate music models (Briot et al., 2020). Intuitively, it measures the difference between two probability distributions: the target distribution (the actual music sequences), and the predicted one (the model's predictions of these sequences). The exponentiation of the entropy, known as perplexity, brings the metric to a linear scale which makes it easier to interpret.

Evaluating the predictive abilities of a music model concerns several aspects and properties of music, including music structure. This last property becomes more evident with longer

Model	Cross Entropy	Perplexity	No. of parameters
Basic LSTM	3.42	148.4	1420K
Attention LSTM	3.31	107.8	1521K
Transformer	3.08	109.5	1001K
UT	3.07	151.3	540K
LSTUT	**2.44**	**61.8**	1308K

Table 5.6: Music modelling evaluation on the test set.

sequences, thus pushing a model to make better use of its parameters rather than increasing its complexity. More precisely, a model that learned to predict music sequences of considerable length (e.g. with 5K+ tokens) has potentially learned to relate musical patterns at different time scales rather than memorising all sequence elements. A possible way to do this is by detecting *repetitions* and *variations*, which are key elements of music structure.

5.3 Results and analysis

Models were trained to minimise the cross-entropy loss with label smoothing set to 0.1 and gradient norm clipping at 0.1. We used the Adam optimiser with the learning rate schedule from the original transformer paper (Vaswani et al., 2017), together with an early stopping policy (with 20 epochs of patience, and a delta of 10^{-3}) to prevent over-fitting.

5.3.1 Learning to predict music

The results pertaining to the evaluation of the models are shown in Table 5.6.

From the results of a Kruskal-Wallis H-test, we found that the distributions of the cross-entropy evaluations associated to the five music models under analysis differ significantly ($\chi^2 = 682.24$), with p-value less than 0.0001. Post-hoc multiple comparisons (Kolmogorov-Smirnov tests) were then performed to detect significant differences between each pair of models (Bonferroni corrections were applied to account for multiple comparisons). From these tests, we found that only the cross-entropy evaluations of the Transformer and the UT were not statistically significant ($p = 0.61$), whereas those of all the others were ($p < 0.05$).

Results show that the LSTUT achieved the best music modelling performance, and statistically outperformed all other models. The use of an attention mechanism in the LSTM improved its modelling performance, thereby confirming previous results (Waite, 2016). All the transformer-based networks outperformed the LSTMs, with the UT achieving comparable results to the vanilla transformer with only half of the parameters.

5.3.2 Learning music structure features

To analyse the extent to which the LSTUT can learn features related to music structure, we manually inspected the attention distributions of its attention heads on a given piece of music. The attention distributions are then compared to the self-similarity matrix (SSM) computed on the same piece. As introduced in Section 2.2, SSMs are a powerful tool extensively used in music structure analysis. To compute an SSM, the music signal is divided into a sequence of feature vectors and all elements of the sequence are compared with each other for similarity. When visualising an SSM, repetitions typically yield *path-like structures*, whereas homogeneous regions yield *block-like structures* (Müller, 2015).

For this analysis, we chose "Fly" by Ludovico Einaudi – a minimalist composition for piano. This contemporary music piece is well suited for our analysis due to its rich and diverse structure. The SSM of this track is shown in Figure 5.8, highlighting the presence of both local and global structural components. This is complemented by the visualisations of a selection of the attention heads of our model, obtained when the track is autoregressively fed to the network.

5.4 Chapter summary

This chapter introduced the music modelling framework (MOlliE), a suite of computational methods and tools providing a standardised methodology for the systematic evaluation of music models. As extensively discussed in previous sections, this is fundamental to ensure comparability of methods for music modelling, and to inspect their behaviour before bringing them to any application domain (e.g. automatic music transcription, computer-assisted composition, etc.). Currently, MOlliE supports a number of symbolic music encodings, state-of-art neural network architectures, and visualisation utilities allowing experimenters to design, evaluate and compare their music models in a controlled environment. The framework also includes the *long-short term universal transformer* (LSTUT), a novel architecture combining recurrence and attention paradigms, and integrating computational improvements to handle long sequences.

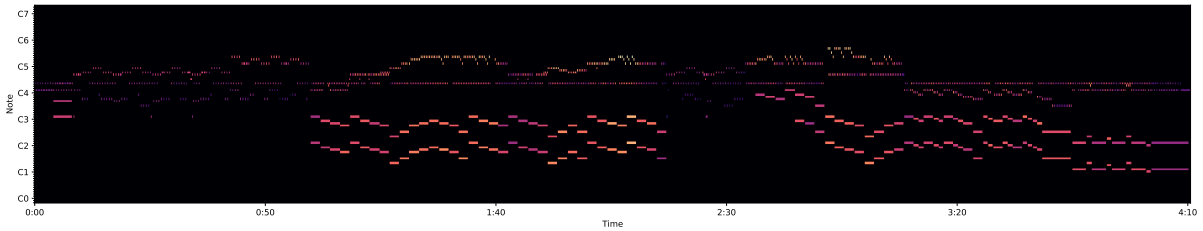


Figure 5.7: Pianoroll representation computed from the MIDI track – an alternative way to visualise the structure of the composition.

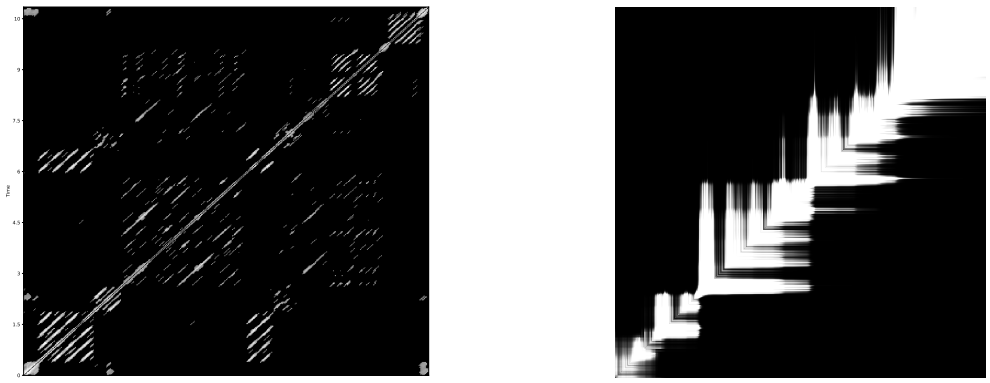


Figure 5.8: A SSM of "Fly" (*Left*) compared with the attention distribution of a selected head in the LSTUT (*Right*), demonstrating that structural features are learned.

Overall, the proposed framework allows to conduct music modelling experiments in a controlled environment promoting the reusability of architectural components and symbolic encodings, while fostering the comparability of music models under the same evaluation criteria and analytical tools. To address RQ 1, the ablation study carried out with MOlliE tested a number of music modelling architectures (variations of LSTMs and Transformer architectures) based on three modelling paradigms as ablation dimensions: *recurrence*, *inductive bias* (for non-recurrent architectures), and *attention*. This was done under an experimental setup specifically designed to provide a challenging testbed for learning long-term dependencies from music sequences in symbolic format: the prediction of transcribed performances of classical music pieces spanning over several minutes, hence pushing music models to make better use of their memory capacity (e.g. detecting repetition and variation, as opposed to full memorisation).

First, the experimental methodology confirmed the utility of MOlliE as a framework for music modelling experiments – from the data exploration and processing stages, to the modelling and the evaluation steps outlined in Section 4.1 (and illustrated in Figure 5.1). In addition, the results demonstrated that architectures implementing all the three modelling paradigms (the LSTUT) can significantly outperform all the other architectures (those based on one or two paradigms) on the polyphonic music modelling task. Based on our assumptions, this indicates that all modelling dimensions were found to play a pivotal role in improving the model's ability to learn structural features from music. A qualitative analysis of the attention distributions of the LSTUT on a structurally complex piece of music also corroborated this finding. Indeed, some attention maps were found to consistently align with the harmonic-melodic structures emerging from the self-similarity matrix (SSM) computed on the same piece (following the extraction of chroma features from the synthesised score). Together with the previous results, this supports

the hypothesis that the LSTUT can learn features related to music structure.

To conclude, the experiments reported in this chapter focused primarily on testing the capabilities of music models to learn long-term dependencies in music – the main scope of RQ 1. On the other hand, generations sampled from these models were found to still require more research effort. This implies that learning multiple music properties is still an open problem, and a gap between modelling music and generating original and realistic compositions from the former models exists. Nonetheless, a model failing to learn long- and short-term structure cannot be expected to generate music with these properties, which are instead peculiar to real compositions. In this regard, focusing on the structural evaluation of music models at this stage provides an upstream approach to mitigate the problem from the main source.

Chapter 6

Unveiling the hierarchical structure of music

To start addressing the second research question (RQ 2), we first introduce a novel method for hierarchical music structure analysis (MSA) – an algorithm that can detect structural components at different levels of granularity, and thus segment (or decompose) a given track as a tree of progressively refined structures (see Section 2.2 for an overview of the MSA task). Our method, named *musical structure communities* (MSCOM), addresses the music segmentation task as a community detection problem. First, a given music piece is partitioned into a number of consecutive audio frames, and an indirect graph reflecting both local temporal connectivity and long-term recurrence information is generated from these frames (Section 6.1.1). The graph is then processed by a divisive community detection procedure based on modularity optimisation (Newman, 2004a) at multiple resolution levels (Arenas, Fernandez, & Gomez, 2008), which yields a structural hierarchy whereby the first level contains a single segment embracing the whole piece and the deepest level contains as many segments as the number of frames (Section 6.1.2). We test and validate MSCOM on the structural annotations for large amounts of music information (SALAMI) dataset (J. B. L. Smith et al., 2011) (Section 6.2). By comparing our results to state of the art methodologies for hierarchical segmentation, we demonstrate that our method outperforms the current approaches. Furthermore, we show that MSCOM can also enable the visualisation and the analysis of musical structure at finer levels of detail, whereby tree representations of music (Lerdahl & Jackendoff, 1985) can be further enriched to reflect more structural relationships (e.g. Lamere, 2000; Wattenberg, 2000).

This algorithm provides the basis for the automatic analysis of structural complexity from audio signals, due to the unsupervised nature of the procedure and the richness of the estimated hierarchical segmentations, which are particularly suitable to be further analysed. The work in this chapter was presented in (de Berardinis, Vamvakaris, Cangelosi, & Coutinho, 2020).

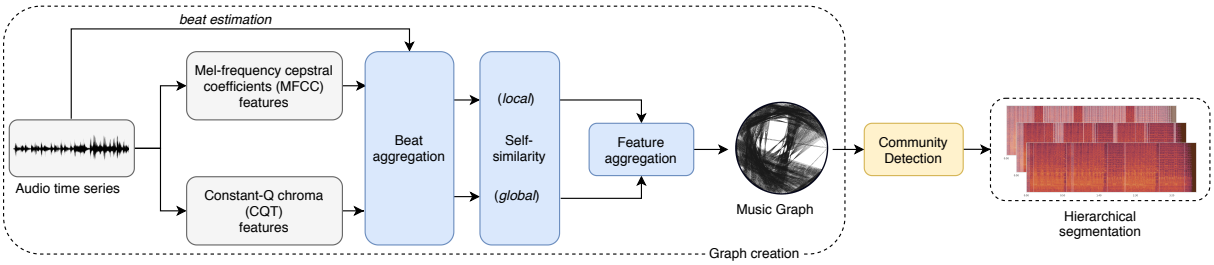


Figure 6.1: Schematic overview of MSCOM with all the main steps of its workflow.

6.1 The musical structure communities (MSCOM) algorithm

Our approach for hierarchical segmentation of music inherits some technical aspects from the LSD method outlined in the previous chapter (Section 2.2.2), and it relies on the intuition that human-composed music has the kind of modular structure that can be uncovered by using multi-resolution community detection algorithms. By exploiting the method of McFee and Ellis (2014a) for the creation of a graph encoding both the temporal and the similarity dependencies among sample frames of a musical piece, we apply a hierarchical clustering procedure in order to detect structural communities at different resolution levels from the so-obtained graph. To emphasise the technical nature of the segmentation approach, we name our algorithm *musical structure communities* (MSCOM). A schematic illustration of MSCOM is shown in Figure 6.1 and a detailed technical description is provided in the following subsections.

In contrast to supervised methods based on convolutional neural networks (Grill & Schlüter, 2015), which can estimate fixed-depth segmentations based on the specific annotation levels provided by a structurally-annotated dataset, our algorithm is fully unsupervised. According to the taxonomies proposed by Peeters (2003) and Paulus et al. (2010), MSCOM can be categorised as a homogeneity- or state-based procedure, identifying structural segments based on the integrity of their acoustic features. Conversely to HMM-based methods Paulus (e.g. 2010), MSCOM does not assign functional labels (e.g. verse, chorus) to the estimated segments. To contextualise our contribution, it is also worth to observe that our algorithm is not the first approach employing community detection techniques in MIR. More precisely, Serrà, Zanin, Herrera, and Serra (2012) investigated the use of community detection algorithms to improve the performance and the interpretability of cover identification methods. Gulati, Serra, Ishwar, and Serra (2016) applied community detection on structural segmentations of Indian music to characterise the discovered patterns into rāga, composition-specific and gamaka motifs.

6.1.1 Music graph construction

The definition of the music graph that we use as part of our methodology is based on the work by McFee and Ellis (2014a). In particular, we create two distinct graphs capturing different perceptual aspects of a musical piece, and a combination of them is considered in order to obtain a single graph to further process. First, two contrasting audio descriptors are extracted from a raw audio file – *harmonic features* (chroma features), for detecting long-range repeating forms, and *timbral features* (mel-frequency cepstral coefficients), for detecting local consistency (see Section 2.2.1 for an overview of audio features used in MSA). To reduce data dimensionality and remove transient noise, both features are beat-synchronised by averaging all the vectors belonging to the same estimated beat. This yields two feature matrices $P = \{p_1, p_2, \ldots, p_N\} \in \mathbb{R}^{d_p \times N}$ and $M = \{m_1, m_2, \ldots, m_N\} \in \mathbb{R}^{d_m \times N}$ denoting respectively the beat-synchronised chromagram and the sequence of mel-frequency cepstral coefficients (MFCC) with corresponding dimensions d_p and d_m and length equal to N, the number of estimated beats.

From the beat-synchronised feature matrices, we compute their self-similarity matrices S^P and S^M, both of dimension $N \times N$, with a similarity function (e.g. a Gaussian kernel) over feature vectors, s.t. $0 \leq S^F_{ij} \leq 1$ denotes a non-negative affinity between feature vectors f_i and f_j. Since similarity is maximal between identical frames, we zero out the main diagonal of both matrices. As outlined in Section 2.2.1, MFCC and chroma-based features are both related to human perception of musical structure. Hence, combining them into a single representation yields a rich and informative descriptor that allows to detect structural patterns from different musical properties of a given track. The obtained feature matrices allow to define the recurrence graph R and the proximity graph Δ as follows:

$$R_{ij}(P) = \begin{cases} S^P_{ij} & p_i, p_j \text{ are mutual k-nearest neighbours} \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

$$\Delta_{ij}(M) = \begin{cases} S^M_{ij} & |i-j| = 1 \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

where $k > 0$ parameterise the degree of connectivity. With this approach, beat-synchronised feature vectors are considered as nodes in the graphs, and the strength of each connection (according to the constraints imposed above) depends on the similarity between the associated nodes. In Figure 6.2, the recurrence graph captures harmonic and melodic repetitions in a given track, whereas the proximity graph preserves the sequential nature of music by connecting consecutive nodes according to their timbral consistency. The weight of a link connecting two nodes in the proximity graph is equal to the similarity between their MFCC feature vectors if

the nodes correspond to temporally consecutive beats. Conversely, a couple of nodes associated to non-consecutive beats receives a 0-strength link. This formulation is in line with the roles of temporal proximity and similarity in human perception of musical structure (Temperley, 2004).

A smoothing filter is then applied to R in order to suppress uninformative links and promote local connectivity in the graph by filling those gaps which can blur potential repetitions. This is a common enhancement strategy for SSM, as it emphasises repeated structures which would then appear as strong diagonal stripes. The smoothing filter is implemented by using a *windowed majority vote* to each diagonal of R as:

$$R'_{ij} = R_{ij} \cdot maj\{R_{i+t,j+t} > 0 \mid t \in \{-w, -w+1, \ldots, w\}\}, \tag{6.3}$$

where $w > 0$ controls the minimum length for a repetition sequence in terms of number of beats.

As a final step, a single graph representing the piece is obtained by combining R and Δ. Since we need to guarantee that the proximity connections are not excessively outnumbered by the repetition connections (it would be difficult to partition the resulting graph into contiguous segments otherwise), the combination is parameterised by a factor $\mu \in [0, 1]$ as follows:

$$G^{\mu}_{ij} = \mu R'_{ij} + (1-\mu)\Delta_{ij}. \tag{6.4}$$

Since McFee and Ellis (2014a) aim for a process that on average moves either in sequence or across repetitions with equal probability, μ is set to a value that lets the combination assign equal weight to the local and repetition edges. To summarise, the so obtained graph G^{μ} is considered as an adjacency matrix, where G^{μ}_{ij} denotes the similarity of the beat-synchronised audio features corresponding to nodes i and j. This graph is the starting point of our musical structure analysis procedure outlined next.

6.1.2 Community detection of structural patterns

The graph representation of a musical piece is then leveraged to detect structural components (or segments) at different resolution levels. Intuitively, each segment can be considered as a *group* collecting nodes with similar features, where the propensity of a node to be part of a certain group depends on the resolution level chosen at a certain stage of the analysis. In the domain of network analysis, groups are better known as communities. In the context of this work, each community corresponds to a structural component, collecting nodes with homogeneous musical properties at a certain resolution level.

From Figure 6.2, and most evidently from the recurrence graph in the *top-right* plot, we can identify different square-shaped structures varying in size. From a visual analysis, each square can be associated to a community and we can easily notice their nested nature, – communities

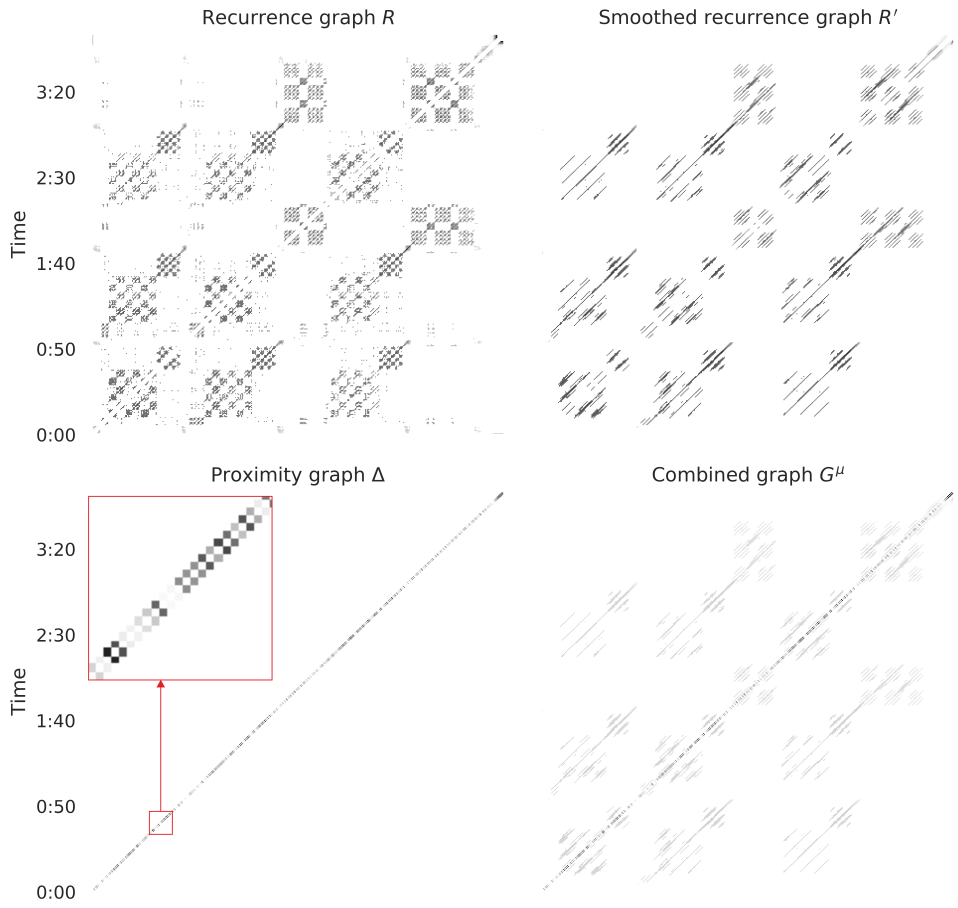


Figure 6.2: The main steps for the creation of the music graph for the track "SALAMI 676" (c.f. Section 6.1.1). The recurrence graph R computed on the chroma features and its smoothed version R' to enhance diagonal stripes are illustrated in the top quadrants. The bottom-left plot represents the proximity graph Δ with a zoomed area highlighting its upper and lower off-diagonals that ensure the linkage of nodes corresponding to temporally consecutive feature vectors. The graph G^{μ} in the last quadrant is a weighted sum of R' and Δ as outlined in Equation 6.4.

being parts of larger communities, which is related to the hierarchical structure of music form (Section 2.2). Identifying these communities at different levels enables us to detect musical structures at different resolutions or scales. On a coarse level, the example track illustrated in Figure 6.2 in relation to its graph representation, seems to exhibit a sectional structure corresponding to the ABABCABC form. Ideally, in the higher levels of the resulting hierarchical segmentation, sections A, B and C would correspond to three different communities, which will be recursively decomposed to detect structural patterns of finer resolutions. Unfortunately, this implies that our method would not be able to detect strophic forms (e.g. AA, AAA).

Community detection is a challenging task in graph theory, since neither the size nor the number of communities is known in advance. The goal is to partition a graph into a number

of communities such that the intra-community degree of similarity is higher than the inter-community level of similarity. Community detection techniques have particularly gained popularity in the domain of computer vision, and more specifically for unsupervised image segmentation (Camilus & Govindan, 2012; Mourchid, El Hassouni, & Cherifi, 2016).

Definition of modularity

Since the definition of community is rather abstract and a graph can be partitioned in several different ways, Newman (2004a) proposed the notion of "meaningful communities". According to his proposal, a community is meaningful if the number of intra-community edges, along with their weights, is considerably different than those being expected in a randomly linked set of nodes with similar properties. To quantify the meaningfulness of a partition for a given graph, Newman (2004a) introduced a quality function called *modularity*, which takes values in the interval $[0,1]$ (with higher values indicating a higher quality partition) and is defined as:

$$Q(W,C) = \frac{1}{2w}\sum_i\sum_j (w_{ij} - \frac{w_i w_j}{2w})\ \delta(C_i,C_j) \tag{6.5}$$

where W is the adjacency matrix of a graph, C is a given partition of the graph into communities, w_{ij} is the weight of the link connecting node i to node j, $w_i = \sum_j w_{ij}$ is the strength of node i, $2w = \sum_i w_i$ is the total strength of the network, C_i is the community to which node i belongs, and $\delta(C_i,C_j)$ is the Kronecker delta function which returns 1 if nodes i and j are members of the same community, 0 otherwise. In case of an unweighted graph, i.e. $w_{ij} \in \{0,1\}\forall i,j$, we replace the strength w_i of node i with its degree d_i (the number of edges adjacent to node i). However, the calculation of modularity is not applicable when the graph is not strongly connected, although it could be done separately for each connected component. This condition further motivates the use of the proximity graph, which ensures the strong connectedness of G^μ.

A partition C of W is optimal if it maximises $Q(W,C)$, though modularity optimisation is an NP-hard problem and the computational cost of any brute force exploration is prohibitive for graphs larger than a few tens of nodes (Duch & Arenas, 2005). To mitigate this problem, heuristics-based approaches for modularity maximisation have been proposed in literature (Clauset, Newman, & Moore, 2004; Duch & Arenas, 2005; Newman, 2004a, 2004b; Pujol, Béjar, & Delgado, 2006). Nonetheless, modularity has a natural resolution limit (Fortunato & Barthelemy, 2007), meaning that vanilla modularity optimisation cannot reveal nested communities beyond a certain resolution level. To overcome this problem, Arenas et al. (2008) introduced a method revealing the community structure of a graph at different resolution levels by manipulating the total strength $2w$ of a given graph. From a technical perspective, increasing the total strength $2w$ without affecting the main structural properties of a network is achieved

through the introduction of self-loops with weights equal to r for all nodes in the original adjacency matrix. The adjacency matrix is then written as $W_r = W + rI$, were W is the original adjacency matrix, r defines the granularity level and I is the identity matrix. Hence, r can be thought of as a *resistance parameter*, defining the tendency of nodes to form communities as it allows to leverage the relative importance of any link in the graph: when $r = 0$, we only access community structures within the resolution limit; when $r < 0$ we can reveal *super-structures* beyond the resolution limit, since nodes are more reluctant to form small scale communities; when $r > 0$ we increase the importance of any individual link so as to reveal *sub-structures*.

According to Arenas et al. (2008), the approximate value of r s.t. all nodes belong to a single community is $r_{min} = \frac{-2w}{N}$, where N is the number of nodes. In this setting, the average strength of all nodes is zero and the weight of the expected edge between any pair i, j increases so that any non-trivial community would appear to be no different from a randomly drawn network. All nodes are thus assigned to one community. Instead, r_{max} is the smaller positive number r that satisfies $w_{ij} < \frac{(w_i+r)(w_j+r)}{2w+Nr}$ $\forall i \neq j$, with each node forming a single-member community called *singleton*. In the resulting segmentation tree, singletons thus correspond to *leaf nodes*.

Ensuring hierarchical consistency

Although this approach enables the detection of communities at different levels of granularity, optimising modularity for different values of r does not guarantee a truly hierarchical partitioning of the graph (Lancichinetti & Fortunato, 2011). This means that every partition obtained by maximising $Q(W_r, C)$ for a certain value of r is independent from any other partitions of the same graph for a different r. The same issue arises when using the LSD method (McFee & Ellis, 2014a), resulting in multi-level segmentations which are not guaranteed to be strictly hierarchical (Section 2.2.2). As clarified later, this violates the *monotonicity* property.

Interestingly, the approach proposed by Granell, Gomez, and Arenas (2012) implements a hierarchical procedure that extends the multi-resolution method of Arenas et al. (2008) in order to ensure a tree-like structure of the obtained hierarchies. As outlined in Algorithm 1, we used this procedure to hierarchically segment a music graph G^{μ} without imposing any constraint that could bias or limit the detection of communities, in terms of their number and type, as well as the depth and topology of the resulting hierarchy.

The community detection procedure starts by setting r to r_{min}, s.t. the modularity optimisation process distributes all nodes to the same community. An iterative process is then repeated by increasing r until we obtain an optimal partition where every community becomes a singleton. At every step, we iterate over all communities identified at the previous step and we run the modularity optimisation algorithm on each community, separately, using the current value

Algorithm 1 Hierarchical community detection

Given the $N \times N$ adjacency matrix W of a graph
Given Δr, a fixed step increment for r
Let W[S] be the square sub-matrix obtained by selecting the rows and columns of W with index in S

$\ell \leftarrow 1$
$r \leftarrow \frac{-2w}{N}$
$W \leftarrow W + rI$
$C^{\ell} \leftarrow \{C_1^{\ell} = \{1, 2, \ldots, N\}\}$ ▷ all node indices in C_1^{ℓ}
while $|C^{\ell}| < N$ **do**
    $\ell \leftarrow \ell + 1$ ▷ current level
    $C^{\ell} \leftarrow \{\ \}$
    for $C_j^{\ell-1}$ **in** $C^{\ell-1}$ **if** $|C_j^{\ell-1}| > 0$ **do**
        $P_{C_j^{\ell}} \leftarrow \{C_{j,1}^{\ell}, C_{j,2}^{\ell}, \ldots, C_{j,m}^{\ell}\}$ = optimal partition of $W[C_j^{\ell-1}]$
        $C^{\ell} \leftarrow C^{\ell} \cup P_{C_j^{\ell}}$
    end for
    $r \leftarrow r + \Delta r$
    $W \leftarrow W + rI$
end while

of the resistance parameter r. This approach ensures a strictly hierarchical community structure which allows to segment music in a consistent manner. At the end of every iteration, we increase r by Δr so that we can progressively access communities of finer granularity.

The increments of r are domain specific and hyper-parameter tuning techniques are necessary. In the context of this work, we measured the degree by which communities split into sub-communities as we increase r, and we found that increments of $\Delta r = 0.1$ enable to access meaningful communities at various resolutions. Values higher than 0.1 lead to information loss due to unidentified communities between consecutive levels in the hierarchies, while smaller values lead to minor variations of communities between successive levels. Since we are still segmenting music from graph representations, as similarly done by the LSD method, segment boundaries are inferred from the changes in cluster membership between consecutive nodes.

6.1.3 Two variants of MSCOM: baseline and dynamic

As outlined in the previous subsections, our methodology is organised in two steps: (i) construction of the music graph from a given track; and (ii) hierarchical multi-resolution detection of communities from the resulting graph. Whereas the second part is left unchanged, we implemented two versions of MSCOM:

- *MSCOM baseline* (MSCOM): makes use of the graph construction procedure of McFee and Ellis (2014a), as explained in Section 6.1.1;

- *MSCOM dynamic* (*DMSCOM*): based on a slightly different procedure where the self-similarity matrix S^P computed on the beat-synchronised chroma features undergoes a dynamic filtration step before the construction of the recurrence graph R as in Equation 6.1; more precisely, depending on the total strength of the network, we filter out those edges in S^P that do not meet the following condition:

$$S^P_{ij} > \lambda^{-1} \frac{\sum_{i'=1}^{N} \sum_{j'=1}^{N} S^P_{i'j'}}{N(N-1)} \tag{6.6}$$

  where N is the number of nodes in the graph, the numerator is the capacity of the network of a certain track, the denominator is the maximum possible capacity of the network, and λ is a positive constant to control the severity of the filtration.

The fraction above can be considered as the average weight of a link in the graph, with the value of λ^{-1} defining a relative threshold for retaining structurally meaningful connections. We found that $\lambda = 4$ enables a reasonable filtration for the dataset considered in the evaluation section. An adequate strategy for tuning this hyper-parameter is to identify the value of λ maximising the clustering coefficient of the resulting filtered graphs. This simple method can also be considered for each segmentation.

6.1.4 An example of hierarchical segmentation

To provide a practical example of the proposed methodology, we briefly report the structural segmentation obtained with MSCOM on the ”SALAMI 676” example track (“The Loner” by Neil Young). As can already be seen from Figure 6.2, the long-term structure of this rock song clearly emerges from the block-like structures in the recurrence plot. A closer visual analysis also reveals the presence of nested sub-structures corresponding to progressively shorter musical ideas. These structural patterns at different resolution levels are what we aim to detect with the hierarchical multi-resolution community detection procedure of MSCOM.

As shown in Figure 6.3, it is possible to visualise the nested structures in terms of communities formed throughout the hierarchical segmentation performed by MSCOM on the track under analysis[1]. As noticeable from the chart, communities do not split homogeneously as we traverse the hierarchy: some groups are more resistant to fragmentation, and they can persist

[1] Although each colour should uniquely corresponds to a community, and viceversa, some of them are repeated in the diagram due to the limited number of colours available in the chosen colour map

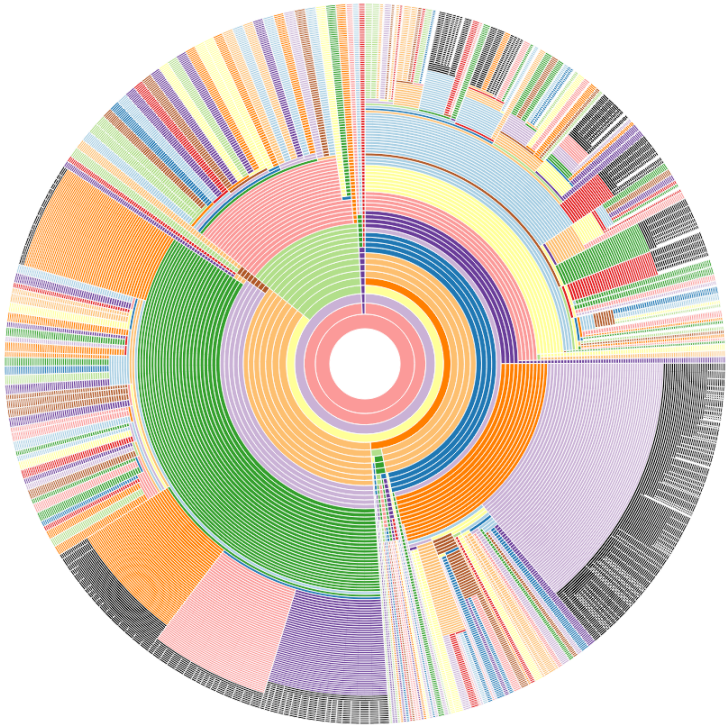


Figure 6.3: Hierarchical segmentation of "SALAMI 676" with colours identifying communities. The innermost circle corresponds to the first segmentation level in the hierarchy, where all nodes belong to the same community, i.e. the starting point for the community detection algorithm; conversely, every node in the outmost circle forms a community per se. This plot is thus helpful to visualise the structure of the detected communities by illustrating how they progressively break into smaller ones.

until the deepest levels in the hierarchy, whereas others tend to break rather quickly. Moreover, the fragmentation ratio of communities from one level to the next one is also related to the structural complexity of each group. In fact, if a certain community corresponds to a segment which does not contain any further sub-structures, then it would fragment into several smaller communities, which in the extreme case would be as many as the number of nodes in the former group. These observations will be crucial to finally address RQ 2 in Chapter 7.

As detailed in Section 6.2.1, the SALAMI dataset provides two levels of reference structural annotations for each track: an upper-level segmentation, where the structural patterns typically corresponds to sections, and a lower-level segmentation that can reach the phrase level. To show the behaviour of MSCOM on the track under analysis, we compare both these human annotations with the segmentations estimated by the algorithm at the levels with closest alignment. In particular, Figure 6.5 compares the upper-level human annotation (the rightmost plot) with the estimation at level 18, for which we observe a distinct overlap; considering the good level of

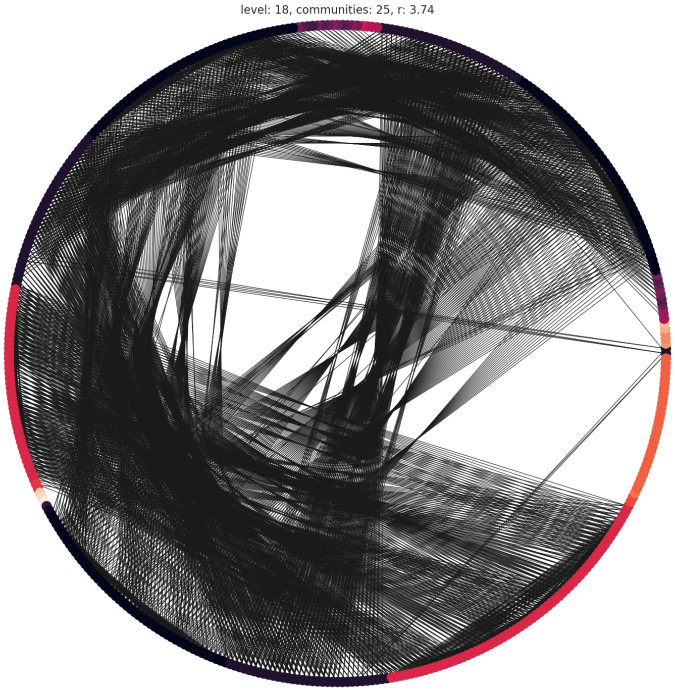


Figure 6.4: The combined graph obtained from SALAMI 676 (the same example track for which its adjacency matrix is plotted in Figure 6.2) with nodes coloured according to the segmentation of the network at the 18th level (plotted in Figure 6.5 and compared with human structural annotations). Considering the inheritance from the proximity graph, nodes are connected as a chain to reflect the temporal dimension of music whereas edges connecting non-subsequent nodes denote their harmonic and melodic similarity.

alignment, we also report the state of the music graph in Figure 6.4. The additional segmentations reported (at levels 4 and 16) demonstrate that our algorithm can also detect superstructures which are more general than sections. Nonetheless, a closer analysis is needed to assess whether any correspondence with the manual identification of these structures can be found.

Analogously, Figure 6.6 presents a comparison of the lower-level human segmentation with the segmentation estimated by MSCOM at the 59th level. From a visual inspection we can notice that reference and estimated segmentations are similar with each other, even tough some structures are more fine-grained in the estimations. A reasonable explanation for this behaviour is related to the fact that communities do not split homogeneously along the hierarchy, thus a closer alignment with the reference segmentations would be obtained by combining multiple segmentation levels together and enforcing some properties during the creation of communities. In doing so, we would reduce the premature fragmentation of communities and ensure a more balanced split of musical structures across the hierarchy. However, we preferred to avoid any a posteriori manipulation that could bias the detection of structural patterns by modularity maximisation, and provide instead a parameter-free algorithm for structural analysis of music. We

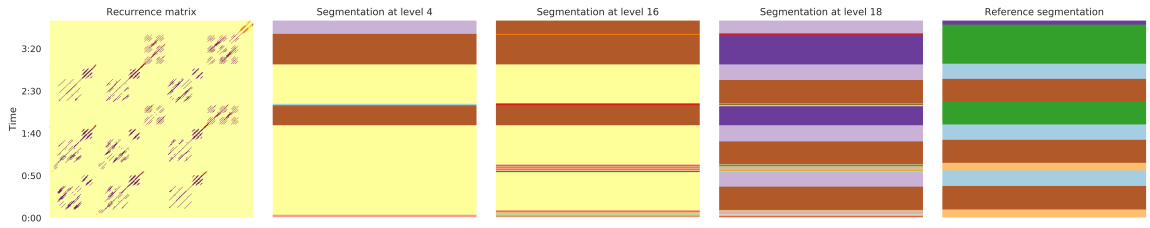


Figure 6.5: Comparison of the upper-level human segmentation for "SALAMI 676" (the rightmost) with the estimated segmentation at levels 4, 16, 18 in the hierarchy produced by MSCOM. Whereas the structural segmentation at the 18th level is at the same granularity of the upper-level reference annotations, the segmentations at levels 4 and 16 uncovered superstructures.

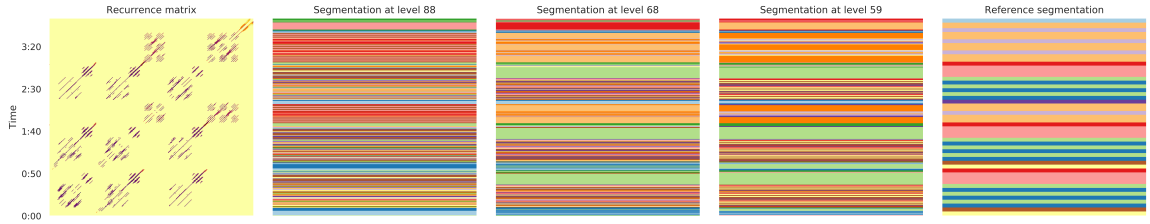


Figure 6.6: Comparison of the lower-level human segmentation for "SALAMI 676" (the rightmost) with the estimated segmentations at levels 88, 68, 59 in the hierarchy produced by MSCOM. Analogously, whereas the 59th segmentation level has the closest alignment with the lower-level reference annotation, those at levels 68 and 88 enables the discovery of more granular structures, providing a deeper analysis of the structure of the piece.

also report two segmentations deeper in the hierarchy in order to show the ability of MSCOM to detect structural patterns at a finer level than musical phrases, thereby potentially unveiling motifs, i.e. the smallest structural units possessing thematic identity (White, 1984).

6.2 Experimental evaluation

To evaluate and validate our methodology, we designed a comparative experiment for the task of hierarchical segmentation of music with the current state of the art methods outlined in Section 2.2.2 (LSD and OLDA). The experimental design consisted in applying both these methods and the two variants of MSCOM on the SALAMI dataset (Section 6.2.1). Then, a quantitative evaluation of the estimated structural segmentations by means of a consolidated measure in the literature of hierarchical MSA (Section 6.2.2) is carried out for each algorithm. This enables a consistent comparison of these segmentation methods in regard to their ability to detect musical structure, accounting for its hierarchical nature. In the following subsections we describe the experimental setup, including a description of the SALAMI dataset, the evaluation metrics used, and the configuration of the algorithms being compared.

6.2.1 Dataset

We used the SALAMI dataset (J. B. L. Smith et al., 2011), one of the largest collections of structurally annotated tracks for hierarchical (and flat) MSA. In particular, SALAMI includes hierarchical annotations for 1359 music tracks of a variety of music styles (e.g., jazz, blues, classical, Western pop and rock, and non-Western ("world") music). The annotations were manually produced by 8 different human experts, all pursuing graduate studies in either music theory or composition, and contain three levels of segmentations per track: *lower*, *upper*, and *function*. The *lower* level typically corresponds to short phrases, the *upper* one represents larger sections, and the *function* level applies semantic labels to large sections (e.g. "verse" or "chorus"). Since the boundaries of the function level often coincide with those of the coarse level, we do not consider the function level in this experiment.

Data collection and pre-processing

Whereas the structural annotations of the tracks in the SALAMI dataset are accessible from the project website[2], the audio files of most of these tracks are not publicly available due to copyright restrictions. Indeed, the music files used by SALAMI come from 4 distinct sources: Codaich[3]; the Internet Archive's live music collection[4]; the RWC music database[5]; and the Isophonics corpus[6]. Among these collections, only the Internet Archive can be freely accessed as per instructions given by the maintainers of the SALAMI project. In addition, we used the scripts provided by Smith[7] to download and process the audio files of those tracks from Codaich and Isophonics that can be retrieved from YouTube. This process led to the compilation of 780 audio files which were compared to the reference human annotations in the dataset in order to ensure a consistent alignment between tracks and segmentations. After this analysis, we discarded all tracks with: (i) hierarchically inconsistent annotations – where the low level segmentation is less specific than the high level one; (ii) duration different from the one reported in the related human annotation. After these steps we ended up with 737 annotated musical pieces, of which 412 are from Codaich, 29 from Isophonics and 296 from the Internet Archive. A list of all the SALAMI tracks considered for our experiments is made available in the project repository that can be accessed from `https://github.com/jonnybluesman/mscom`.

[2] `https://github.com/DDMAL/SALAMI`
[3] `http://jmir.sourceforge.net/index_Codaich.html`
[4] `https://archive.org/`
[5] `https://staff.aist.go.jp/m.goto/RWC-MDB/`
[6] `http://isophonics.net/datasets`
[7] `http://jblsmith.github.io/Getting-SALAMI-from-YouTube/`

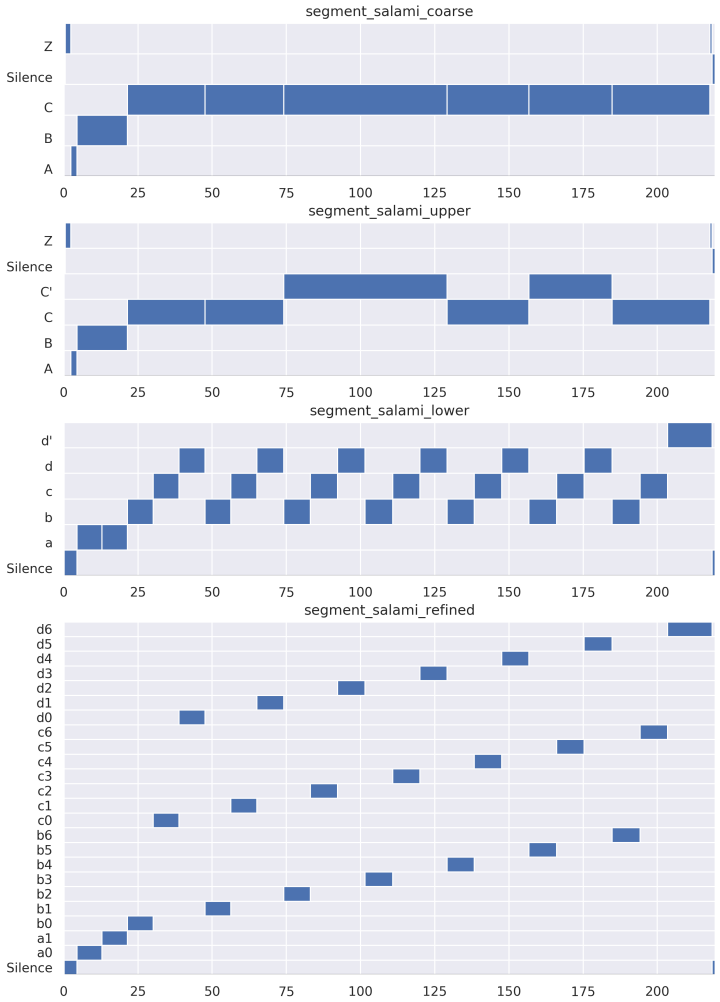


Figure 6.7: Hierachical expansion of the first human annotation of SALAMI 1094. The two segmentation levels denoted with the *upper* and *lower* tags define the original hierarchy, whereas the *coarse* and the *refined* levels are obtained by contracting the *upper* level and refining the *lower* level respectively, according to the expansion method by McFee and Kinnaird (2019).

Automatic hierarchy expansion

As mentioned before, it is common to consider only the *upper* and the *lower* segmentation levels in each reference annotation when evaluating automatic methods for hierarchical MSA. Hence, each reference hierarchy is originally composed of two distinct levels, whereas the hierarchies estimated by the automatic procedures contain several levels of segmentation.

Recently, McFee and Kinnaird (2019) introduced a method for automatically enriching structural annotations by inferring and thus expanding hierarchical information latently encoded in the original segment labels. Given a flat segmentation, their method operates by simultaneously contracting similarly labelled segments – those differing only for a variation marker such as A and A' – and refining segments with identical labels. The so-obtained contraction and

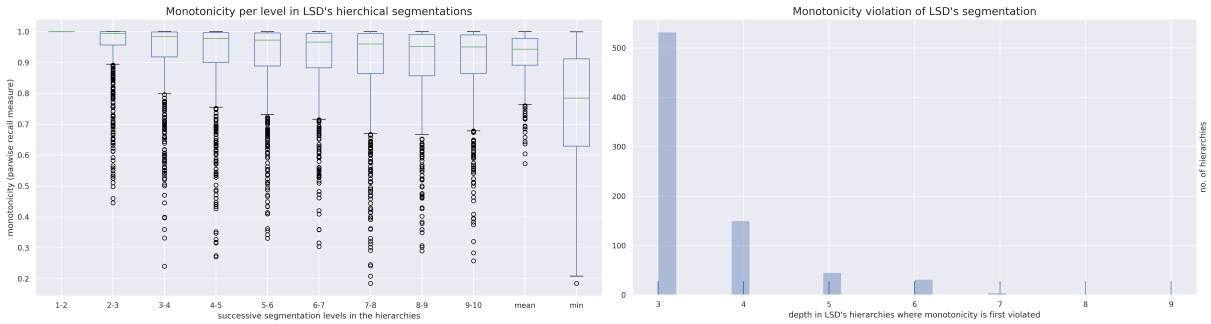


Figure 6.8: Analysis of monotonicity on the hierarchical segmentations produced by LSD. *Left*: distribution of monotonicity for each couple of successive levels in the hierarchies estimated by LSD. *Right*: distribution of the level (or depth) in LSD's hierarchies at which maximum monotonicity is no longer preserved.

refinement levels are combined with the original annotation into a hierarchical annotation: the first level is a contraction of the variation markers (e.g. two distinct segments labelled as A and A' will get the same label); the second level is the original annotation; and finally the third is a refinement of the labels by making each instance of a label unique (e.g. two distinct segments with the same label A would get two unique labels a_i and a_j depending on their occurrence).

By exposing detailed structure latent in the annotations, the expansion method was demonstrated to allow structure comparison methods to more accurately assess similarity between human annotations. Therefore, we believe that this method would also ameliorate the evaluation of automatic methods for hierarchical MSA. For this purpose, we adapted the expansion method to work directly on hierarchical annotations, so as to obtain an extended structural hierarchy where: the first level, called *coarse*, is the contraction of the *upper* segmentation level; the second and third levels are the original annotation and correspond to the *upper* and *lower* levels; the fourth level, called *refined*, is obtained by applying the refinement procedure on the *lower* segmentation level. An example of extended hierarchical annotation is illustrated in Figure 6.7.

6.2.2 Evaluation measures

The following subsections introduce the basic concepts of structural segmentation and provide a technical overview of the hierarchical evaluation measures used for this experiment.

Preliminary concepts

Let $X = \{x_1, x_2, \ldots, x_T\}$ denote a set of sample frames generated from a given track at some fixed resolution, typically corresponding to 10Hz (i.e. 100ms-long frames). A flat *segmentation* S of X is defined by temporally partitioning X into a sequence of labelled time intervals, which

are denoted as *segments*. Formally, a segmentation of X can be encoded as a mapping of samples $t \in [T] = \{1, 2, \ldots, T\}$ to a set of segment labels $Y = \{y_1, y_2, \ldots, y_k\}$, thus defining S as $S : [T] \rightarrow Y$. Depending on the dataset under analysis, Y may consist of functional labels, such as *intro*, *verse* and *chorus*, or section identifiers such as A and B.

A *segment boundary* is any time instant at the boundary between two segments. Let $S(i)$ identify the label of the segment containing the i-th frame in X, it usually corresponds to a change of label $S(t) \neq S(t+1)$ for $t > 1$, though boundaries between similarly labelled segments can also occur (e.g. an AA form). As outlined before, comparing an estimated flat segmentation S^E with a reference one S^R can be done with different measures, evaluating either the agreement of segment boundaries (i.e. boundary detection) or that of segment labels (i.e. structural grouping). Nonetheless, these measures cannot be directly applied on a *hierarchical segmentation*, which is defined as a tree of flat segmentations:

$$H = (S_0, S_1, S_2, \ldots, S_m), \tag{6.7}$$

where each level is a refinement of the preceding layer with the ordering typically implying a coarse-to-fine structural analysis of the corresponding track. A hierarchical segmentation defined in this way is generally expected to be *monotonic*, meaning that a segmentation level S_l would only introduce new segment boundaries to those identified in S_{l-1}, while maintaining the labels of those segments that are not decomposed. Formally, we say that a hierarchy H is monotonic if for every level l, the following condition holds

$$S_l(u) = S_l(v) \quad \Longrightarrow \quad S_{l-1}(u) = S_{l-1}(v). \tag{6.8}$$

This definition of monotonicity is binary, but it can be relaxed by measuring the proportion of time instants u and v where agreement at level l implies agreement at level $l-1$. As observed by McFee and Kinnaird (2019), this is calculated exactly by the pairwise recall measure (Levy & Sandler, 2008), when S_l is considered as the reference and S_{l-1} is the estimated segmentation. A measure close to 1 would express high monotonicity between two segmentation levels S_i, S_j (with 1 denoting full monotonicity), whereas lower values indicate violation of monotonicity.

The L-measure

The L-measure (McFee et al., 2017) is currently the only evaluation metric in the literature that makes it possible to compare two hierarchical segmentations accounting for both the subtasks in MSA. To understand the intuition behind this metric, we need to introduce the concept of *meet* between sample frames. Formally, given a hierarchical segmentation H and sample frames u, v the *meet* of u and v under H is defined as:

$$M(u,v \mid H) = \max\ k \ \text{ s.t. } \ S_k(u) \ = \ S_k(v), \tag{6.9}$$

that corresponds to the deepest level in the hierarchy where the frames u and v receive the same label. Therefore, the *meet* induces a partial ordering over pairs of time instants: large values of $M(u,v \mid H)$ indicate a high degree of similarity and small values indicate low similarity.

To compare two hierarchical segmentations H^R and H^E, the L-measure is based on examining triplets of distinct time instants t,u,v in terms of the pairwise meets $M(t,u \mid H^R)$ and $M(t,v \mid H^R)$. In particular, the *reference comparison set* for a segmentation H is formulated as:

$$A(H) = \{(t,u,v) \mid M(t,u \mid H) > M(t,v \mid H)\}. \tag{6.10}$$

Therefore, $A(H)$ contains all those triplets of sample frames where (t,u) agree at a deeper level than the pair (t,v). Level-independent precision and recall scores can thus be obtained by comparing the size of the intersection to the reference comparison set:

$$\text{L-Precision}(H^R,H^E) = \frac{|A(H^R) \cap A(H^E)|}{|A(H^E)|} \tag{6.11}$$

$$\text{L-Recall}(H^R,H^E) = \frac{|A(H^R) \cap A(H^E)|}{|A(H^R)|}. \tag{6.12}$$

These scores capture the rank-ordering of pairwise similarity between time instants. The final L-measure is computed as the harmonic mean of L-Precision and L-Recall. Rather than asking if an annotation describes two instants (u,v) as the same or different, these scores check whether (t,u) are more or less similar to each other than the pair (t,v), and whether that ordering is respected in both annotations (McFee et al., 2017).

6.2.3 Monotonicity of LSD's segmentations

As mentioned in Section 2.2.2, the hierarchies produced by LSD are not guaranteed to be monotonic, since each segmentation level S_l in a hierarchy H is the result of a separate clustering step where l distinct musical parts are identified. Therefore, it is possible that S_l does not preserve the segment boundaries detected in S_{l-1}. Nonetheless, McFee and Kinnaird (2019) pointed out that the definition of the L-measure is most intuitive when the underlying annotations are monotonic. Monotonicity though is not a strict requirement for the computation of this metric, as the L-measure depends on the maximum level of agreement between a pair of time instants. However, if a hierarchy is not monotonic, then the maximum level of agreement d between two time instants might not be consistent across the hierarchy if monotonicity is not preserved among

each consecutive couple of preceding segmentation levels $(S_0,S_1),(S_1,S_2),\ldots,(S_{d-1},S_d)$. As a consequence, the L-measure computed on non-monotonic hierarchies may lead to counterintuitive results – which is undesirable for the interpretability of our results.

To ensure a thorough comparison between the automatic segmentation methods considered in our experiments, we analysed the monotonicity of LSD's hierarchical segmentations on the SALAMI dataset. For this purpose, we followed the methodology introduced in Section 6.2.2, that is based on the level-by-level computation of the pairwise recall measure. Since the hierarchies produced by LSD consist of 10 segmentation levels, we obtain 9 measures of monotonicity for each hierarchy (Figure 6.8). From this analysis we found that the first two levels are always fully monotonic, whereas the average monotonicity tends to decrease as we move down through the hierarchies. As reported in Figure 6.8, most of the hierarchical segmentations violate full monotonicity already at the 3rd level, and none of them is fully monotonic until the 10th level.

For the sake of the evaluation, considering the previous remarks on the consistency of the L-measure, we cut LSD's hierarchies until their last fully monotonic level and we included them in our comparison together with the original (non monotonic) hierarchies. Each truncated hierarchy $H^t = (S_0,S_1,\ldots,S_l) \subseteq H = (S_0,S_1,\ldots,S_l,\ldots,S_m)$ satisfies the condition:

$$mono(S_i,S_{i+1}) = 1 \; \forall i < l \; \wedge \; mono(S_l,S_{l+1}) < 1$$

where *mono* is the monotonicity function corresponding to the pairwise recall measure. We denote the resulting truncated hierarchies as LSDM, and we can consider this process as a refinement of the LSD method to ensure the monotonicity of the estimated segmentations.

6.2.4 Experiments

Since only 476 (out of the 737) tracks in the SALAMI dataset are double-annotated, we limit our evaluation to the first annotation available for each track in order to make use of the whole database. To improve the ability of the L-measure to assess similarity between structural annotations, we also evaluate the segmentations yielded by each algorithm on the expanded reference hierarchies (as detailed in Section 6.2.1). Our experiments are organised as follows:

1. Segmentation of the 737 SALAMI tracks using the four algorithms under analysis – LSD, OLDA, MSCOM, DMSCOM – in order to obtain a structural segmentation hierarchy for each track and by each method.

2. Truncation of LSD's hierarchical segmentations in order to obtain fully-monotonic hierarchies that we denote as LSDM (Section 6.2.3).

3. Computation of the L-measures between each structural hierarchy estimated by an automatic procedure and the first reference hierarchy of the segmented track. This is done for both the original and the expanded reference hierarchies.

For the experiments conducted with the LSD and OLDA algorithms, we used the implementations that are freely available in MSAF (Nieto & Bello, 2015). As mentioned in Section 2.2.2, since OLDA can only perform hierarchical boundary detection, it is necessary to use an algorithm for structural grouping in order to perform a hierarchical segmentation. Similarly to McFee et al. (2017), we used the 2D-Fourier magnitude coefficients method (Nieto & Bello, 2014) to estimate segment labels since it yields state of the art results in terms of automatic (flat) segment label prediction. For the sake of reproducibility of our experiments, we used the default hyper-parameters suggested in MSAF (Nieto & Bello, 2015), which were optimised by the authors on a collection of structurally annotated datasets (which include SALAMI).

Although the segmentation depends on the quality of the affinity matrix described in Section 6.1.1, (D)MSCOM does not require any extra hyper-parameters to be set for the further detection of community structures. Indeed, the hierarchical multi-resolution community detection procedure does not depend on any particular specification that would considerably alter the result of the segmentation. Nonetheless, the user is free to define the step size for the increase of the resistance parameter r, which is currently set to 0.1 by default. Our method is implemented in Python 3.6[8] and uses `librosa` 0.6.2 (McFee et al., 2015) for audio feature extraction.

6.3 Results and analysis

6.3.1 Performance comparisons between all algorithms

	Original reference hierarchies			Extended reference hierarchies		
	L-measure	L-precision	L-recall	L-measure	L-precision	L-recall
LSD	0.462 ± 0.128	0.394 ± 0.120	0.584 ± 0.150	0.480 ± 0.123	0.420 ± 0.120	0.577 ± 0.143
LSDM	0.301 ± 0.179	0.377 ± 0.158	0.289 ± 0.205	0.309 ± 0.179	0.402 ± 0.158	0.282 ± 0.194
OLDA	0.398 ± 0.101	0.325 ± 0.098	0.536 ± 0.111	0.415 ± 0.097	0.348 ± 0.098	0.531 ± 0.104
MSCOM	0.460 ± 0.112	0.382 ± 0.102	0.600 ± 0.135	0.478 ± 0.105	0.408 ± 0.098	0.593 ± 0.129
DMSCOM	$\mathbf{0.480} \pm 0.111$	$\mathbf{0.403} \pm 0.103$	$\mathbf{0.611} \pm 0.133$	$\mathbf{0.500} \pm 0.104$	$\mathbf{0.430} \pm 0.100$	$\mathbf{0.607} \pm 0.127$

Table 6.1: Overview of the segmentation performance – mean and standard deviation of the L-measures – of each algorithm under analysis with respect to the first reference annotation provided for each track in the SALAMI dataset. The evaluation is performed for both the original (*left*) and the extended (*right*) reference hierarchies.

[8] Python: an open source programming language. Python Software Foundation (www.python.org)

	L-measure		L-precision		L-recall	
	O	E	O	E	O	E
			MSCOM			
LSD	ns	ns	ns	*	ns	ns
LSDM	***	***	***	***	***	***
OLDA	***	***	***	***	***	***
			DMSCOM			
LSD	***	***	***	***	*	**
LSDM	***	***	***	***	***	***
OLDA	***	***	***	***	***	***
MSCOM	**	**	***	***	ns	ns

Table 6.2: Summary of the Kolmogorov-Smirnov tests used to detect statistically significant differences between the algorithms' performance on each evaluation metric. For each measure, 'O' denotes the evaluation performed on the original reference hierarchies, whereas 'E' refers to the extended counterpart. *ns*: not significant, $p > 0.05$; * $p \leq 0.05$; ** $p \leq 0.01$; *** $p \leq 0.001$.

The segmentation performance (L-measure, L-precision and L-recall) for each algorithm and experiment is shown in Table 6.1. From the results of the Kruskal-Wallis H-tests, we found that the distributions of the evaluations associated to the five automatic procedures under analysis differ significantly for all measures, for both the original (L-measure: $\chi^2(2) = 718.37$; L-precision: $\chi^2(2) = 264.07$; L-recall: $\chi^2(2) = 1087.59$) and the extended hierarchies (L-measure: $\chi^2(2) = 770.17$; L-precision: $\chi^2(2) = 281.13$; L-recall: $\chi^2(2) = 1144.48$). The p-values associated to these tests are all less than 0.0001. Post-hoc multiple comparisons (Kolmogorov-Smirnov tests) were then performed to detect significant differences between the performances of the various algorithms on each evaluation measure (Bonferroni corrections were applied to control for family-wise error rate of multiple comparisons). In particular, we statistically compared the performances of both our methods with those of the LSD and OLDA algorithms; this is done individually for each performance measure and the original (O) and extended (E) reference hierarchies.

Results show (see Table 6.2) that DMSCOM performed statistically better than LSD, LSDM and OLDA in all metrics and hierarchy types (original or extended). MSCOM also outperformed OLDA and LSDM ($p < 0.001$ for all comparisons) in all metrics and hierarchy types (original or extended), but it only outperformed LSD in the L-precision measure for the extended hierarchies experiment. Finally, we also compared the performances of MSCOM with DMSCOM. As reported in the last row of Table 6.2, statistical analysis indicates that DSCOM outperforms MSCOM for both the L-measure and the L-precision, but not for L-recall.

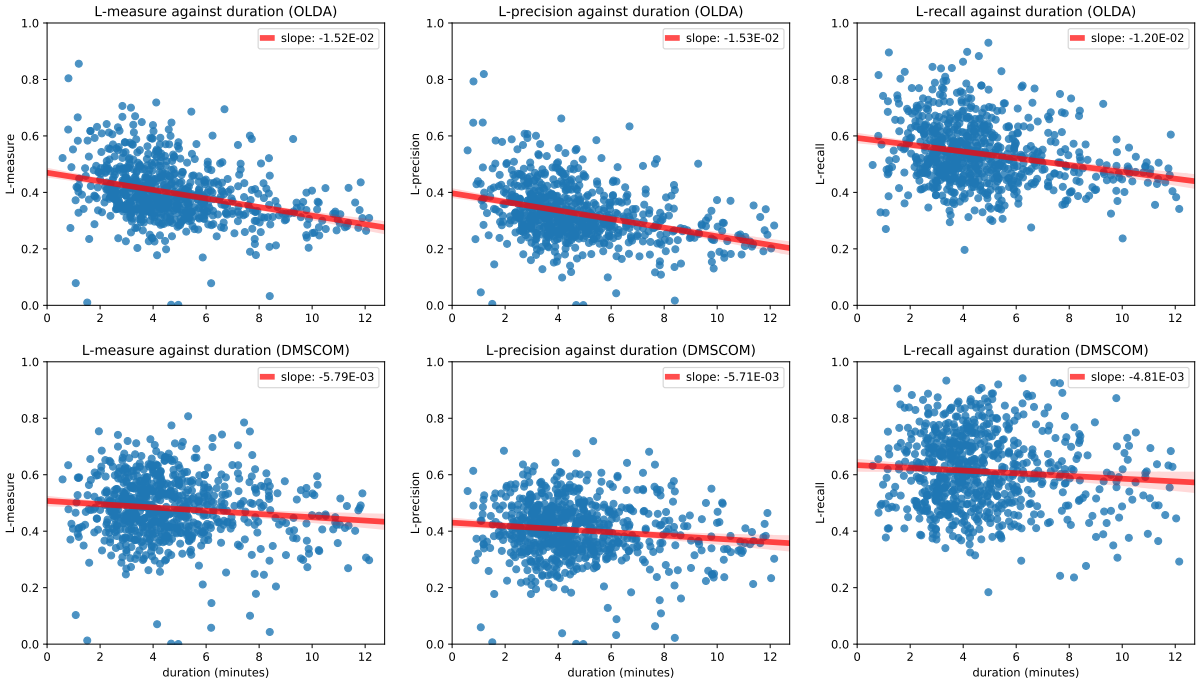


Figure 6.9: Segmentation performance degradation, in terms of the L measures outlined in Section 6.2.2, as function of track duration. The first row reports the trend for the evaluation of OLDA, whereas the second one is related to DMSCOM. A regression line is plotted along with the data to facilitate the comparison of the graphs.

In sum, our results show that DMSCOM has achieved the best results in all evaluation measures and hierarchy types considered in our experiments, and statistically outperformed all other algorithms with the exception of MSCOM on L-recall. MSCOM outperformed both OLDA and LSDM in all performance metrics and for both hierarchy types, but not LSD (with the exception of L-precision on the extended hierarchies experiments, where it performed statistically better).

It should also be noted (see Table 6.1) that the proposed experimental setup suggests that comparing estimated structural segmentations against the extended reference hierarchies (c.f. Section 6.2.1) leads to an increase of L-precision and a slight reduction of L-recall. Given that the former increase is considerable, this translates to a greater L-measure, which is a more accurate estimate of the segmentation accuracy for hierarchical MSA (McFee & Kinnaird, 2019).

6.3.2 Hierarchical depth and performance degradation

As can be seen in Table 6.1, DMSCOM achieves larger average L-recall than L-precision. This result confirms prior observations of McFee et al. (2017) in regard to the tendency of automated methods to identify more detailed structures than were encoded by the human annotators. In

fact, the segmentation algorithms under analysis produce much deeper hierarchies than the reference annotations. Considering that the reference annotations in the SALAMI dataset have fixed depth (see Section 6.2.1), this behaviour may be responsible for lower precision. However, the results obtained from this comparative experiment do not seem to indicate a trade-off between precision and recall as a function of hierarchy depth, as pointed out by McFee et al. (2017). Indeed, the structural hierarchies produced by (D)MSCOM are much deeper than those estimated by OLDA, but still can achieve considerably higher L-precision and L-recall.

McFee et al. (2017) also noticed that another factor impacting on the low segmentation performance of the state of the art methods is their weakness in segmenting longer tracks. On such tracks, the existent algorithms are said to over-emphasise short discontinuities and otherwise label the remainder of the track as belonging primarily to one segment. Indeed, the hierarchical segmentation estimated by OLDA on SALAMI-1112 (one of the longest tracks in the dataset, at 713 seconds) achieves L- measure, precision and recall of 0.309, 0.250 and 0.404 respectively, which are below the average performance of the segmenter for the corresponding metrics (Table 6.1). Instead, DMSCOM does not seem to have the same degradation, and the segmentation of the same piece receives L- measure, precision and recall of 0.534, 0.464 and 0.628. Notably, these results are more in line with the average performance of the algorithm.

To better understand this behaviour, we examined the trend of the evaluation results for both OLDA and DMSCOM as a function of track duration. This analysis is limited to these algorithms as they both estimate monotonic segmentations. As shown in Figure 6.9, the degradation trend of the segmentation performance is more evident (notice the line slope) for OLDA in both L- precision and recall. On the other hand, the same pattern is less evident for DMSCOM. We conjecture that the slightly decreasing trend in the evaluation metrics for DMSCOM might be due to the sparsity of long tracks in the SALAMI dataset used in our experiments. We observe that the performance of our method is less sensitive to the track length. Nonetheless, a closer analysis of this behaviour with more longer tracks available is needed to assess whether (D)MSCOM is insensitive to the duration of tracks.

6.3.3 Inferring segmentation accuracy

As suggested by McFee et al. (2017), the segmentation accuracy of the automatic procedures can be estimated by assuming that the L-measures between different human annotations on the same tracks – i.e. *inter-annotator agreement* – define the upper limit for hierarchical MSA performance. For this purpose, we considered all those double-annotated tracks in SALAMI (476 out of 737), and for each of them we computed the L-measures between the two available human annotations. Results are shown in Table 6.3.

Comparing the L-measure of DMSCOM (0.480/0.500) with that corresponding to the upper

	Original hierarchies	Extended hierarchies
L-measure	0.640 ± 0.198	0.678 ± 0.168
L-precision	0.641 ± 0.197	0.683 ± 0.175
L-recall	0.662 ± 0.2	0.694 ± 0.174

Table 6.3: L-measures for the quantification of inter-annotator agreement: an upper limit for the segmentation performance of the automatic methods.

limits determined from human annotations (0.640/0.678), we can notice that there is still a considerable gap between human and automatic segmentation performance. Nevertheless, the L-recall achieved by DMSCOM (0.611/0.607) is not particularly far from the one corresponding to inter-annotator agreement (0.662/0.694). This result suggests that our method can detect most of the musical structures present in a piece of music, which is an important aspect in regard to the potential practical applications of a segmentation algorithm.

6.4 Chapter summary

In the field of music information retrieval, the automatic detection of structural patterns in music is one of the most challenging tasks considering the current limitations of existing procedures. The problem is further compounded when the analysis of structure has to take into account the natural organisation of musical patterns as nested components in hierarchies: from sections to phrases, motives and so forth. To the best of our knowledge, there are only two algorithms in the literature which can segment music hierarchically, i.e. the ordinal linear discriminant analysis (OLDA) (McFee & Ellis, 2014b) and the Laplacian structural decomposition (LSD) method (McFee & Ellis, 2014a), with McFee et al. (2017) providing an evaluation of these procedures.

This chapter introduced MSCOM and DMSCOM, two novel algorithms that can segment music hierarchically and perform both boundary detection and structural grouping as a single task. Our approach is organised in two steps: a music graph encoding the timbral and harmonic properties of a given track is constructed; and an algorithm for hierarchical multi-resolution community detection is applied on the resulting graph. This enables the detection of structural patterns in music at different resolution levels, which can even be more general (macro-structures) or more specific (micro-structures) than the corresponding human annotations.

To test the algorithms, a systematic and reproducible evaluation of (D)MSCOM was conducted, which enabled the comparison of their performance with OLDA and LSD on the publicly available SALAMI dataset. Our results demonstrate that DMSCOM outperforms the former algorithms and achieves significantly better hierarchical segmentation performance.

We also demonstrated that our method possesses key advantages over existing ones. First,

the hierarchical segmentations yielded by (D)MSCOM are more robust to performance degradation associated with long tracks, a key limitation of existing approaches. Second, unlike existing algorithms, the increased depth of the hierarchies produced by (D)MSCOM does not compromise the precision of the segmenter. Third, (D)MSCOM is almost parameter-free, and this makes it potentially applicable to different styles and genres of music without the need to "tailor" its configuration for an optimal segmentation. Fourth, the hierarchies constructed by (D)MSCOM are deeper than the ones estimated by the other algorithms in the literature, since we do not restrict or limit the size and type of segments to detect nor the topology of the hierarchies. This last property is desirable when hierarchies undergo further steps of analysis.

Hierarchical multi-resolution procedures for the detection of structural communities in music tend to create deep hierarchies. For the sake of MSA, post-segmentation strategies could be devised in order to analyse the generated hierarchies at all levels and make them more compact. Although this would make it easier and more intuitive for a human to inspect and interpret the resulting hierarchies, the segmentations produced by (D)MSOM are rich enough to capture and summarise structural properties of their tracks.

In addition to the (D)MSCOM algorithms, we also proposed a revised methodology for the evaluation of hierarchical MSA algorithms, which leverages the recent methods for hierarchy expansion to enrich the reference annotations. Since this new method exploits the latent hierarchical information encoded in the annotations (and provides a more extensive comparison between estimated and human hierarchies), we envisage that this approach can become a standard for the evaluation of MSA algorithms.

In sum, this chapter introduced methods for hierarchical music structure analysis, providing the methodological basis needed to address RQ 2. Indeed, the next chapter further develops this work and introduces new quantitative measures of structural complexity in music from the estimated hierarchical segmentations. As motivated in Section 1.3, this is a technical challenge of great interest for automatic music recommendation systems, automated music composition algorithms and to the field of musicology (and particularly music analysis).

Chapter 7

Measuring the structural complexity of music

Building upon the previous contribution, this chapter presents a method for the automatic analysis of structural complexity. Our method method leverages computational methods for music structure analysis (MSA) to detect structures and their nested organisation within a composition. The resulting structural segmentation is then analysed and summarised with a set of metrics we devised to formally describe the decomposition process of the identified musical ideas (Section 7.1.2). In lieu of subjectively defining structural complexity, our approach is based on the hypothesis that the former is a latent property that can be captured by a set of metrics. We tested this method on a large dataset comprising music with different degrees of structural complexity, and found that our metrics can explain structural properties inherent to each complexity class (Section 7.2). We also demonstrated and provided examples on how these metrics can be used for evaluating the structural complexity of music (Section 7.3).

The main contributions of this chapter are a set of metrics quantifying structural properties of music, together with a novel evaluation framework for the automatic analysis of structural complexity from music. Overall, these contributions address RQ 2, thereby providing an easy yet effective method for the quantification of structural properties directly from musical objects. The work in this chapter was presented in (de Berardinis, Cangelosi, & Coutinho, 2022).

7.1 From MSA to a set of structural descriptors

The analysis of hierarchical segmentations can reveal insights into the richness and complexity of music structure. As an example, we show in Figure 7.1 how a simple visualisation of structural hierarchies permits visualising structural differences between random, generated and real music. Here, a sunburst diagram is used (as a compact alternative of a dendrogram) to visualise

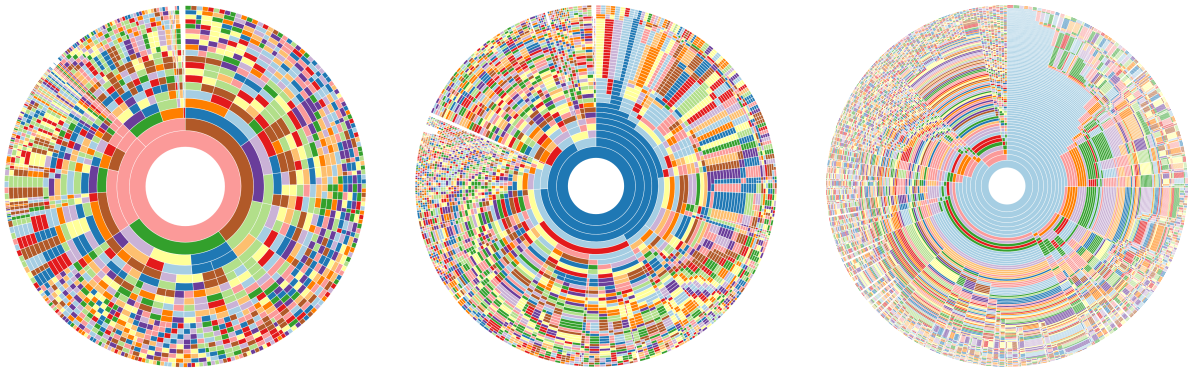


Figure 7.1: Illustration of the hierarchical segmentation of a piece sampled from the distribution of an untrained LSTM (*left*); one generated from an LSTM network trained on a dataset of symbolic classical music (*centre*); and the other chosen from a collection of classical compositions for piano (*right*), all with the same duration. For each plot, the innermost circle corresponds to the first level in the hierarchy, where all audio frames belong to the same segment enclosing the whole piece. From the second level, segments start decomposing into finer structural components (colours denote their identity, although repetitions occur due to their limited availability), until every frame forms a community per se at the bottom of the hierarchy (the outermost circle).

a hierarchical segmentation of a track: from the top level, where all the audio samples are clustered in the same group (the unique slice in the inner-most circle), to the bottom layer, where each temporal fragment of the composition forms its own group (note the full separation in the outer-most circle). This graphical representation of hierarchical structure is the same introduced in Figure 6.3 for the example track considered in Section 6.1.4.

By analysing how music structures progressively break up in a composition, structurally informative descriptors can be used to formalise this process. Our method does so in two steps. First, hierarchical segmentations are computed with the *dynamic musical structure communities* (DMSCOM) algorithm. Second, we derive structural descriptors from the obtained hierarchies.

7.1.1 Structural segmentation of audio music recordings

As detailed in Chapter 6, DMSCOM is a state-of-the-art algorithm that produces rich and deep hierarchical segmentations of music pieces from raw audio. DMSCOM segments music hierarchically and performs both boundary detection and structural grouping as a single task. First, a graph $G = (V, E)$ is constructed from an audio track $X = \{x_1, \ldots, x_T\}$. Nodes correspond to audio samples ($V = X$) and edges encode timbral and harmonic properties of X, with the topology of the network ensuring the connectedness of temporally subsequent samples. Structural segments at different levels of granularity are detected from the music graph G.

In summary, each segment collects nodes with similar features, with the propensity of a node being part of a group (or community) depending on the resolution level at a certain layer

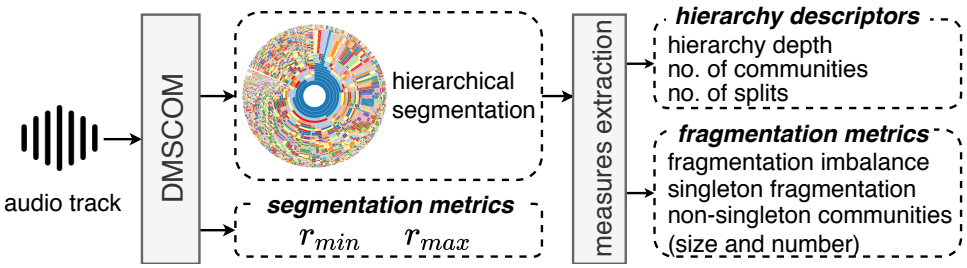


Figure 7.2: Acquisition process and categorisation of the structural metrics.

in the hierarchy. A community thus corresponds to a structural component, collecting nodes with homogeneous musical properties at a certain resolution level. With DMSCOM, this is achieved by using the multi-resolution hierarchical community detection procedure of Granell et al. (2012) on G. The key element of this recursive procedure is the resistance parameter r, used to control the granularity of structures at a certain segmentation level. In particular, self-loops with weight equal to r are introduced for all nodes in the adjacency matrix E in order to control the propensity of nodes forming communities: when $r < 0$ we can reveal super-structures, since nodes are more reluctant to form small-scale communities; when $r > 0$ we increase the importance of any individual link so as to reveal sub-structures. For a more detailed treatment of the algorithm, we invite the reader to consult Chapter 6.

7.1.2 Metrics of structural complexity

In this work, we use a collection of metrics quantifying specific properties of the hierarchical segmentation process to characterise music structural complexity. As outlined in Table 7.1, they were grouped into three categories: *segmentation metrics*, *hierarchy descriptors* and *fragmentation metrics* (see also Figure 7.2 for an illustration of the acquisition flow).

Segmentation metrics. This group of metrics includes parameters related to the segmentation algorithm – the extreme values of the resistance parameter r needed to hierarchically partition a given track (c.f. Section 6.1.2). r_{min} is the smallest negative value of the resistance parameter s.t. all nodes of the music graph belong to a single community. In other words, it relates to the amount of negative "force" that has to be applied to each node in the graph to enclose all of them within the same segment, i.e., how much nodes are resisting to form a single community. r_{max} is the resistance of the music graph to decompose fully into singleton communities – the most atomic structures. It corresponds to the smallest positive r s.t. each node forms a community on its own (*singleton*).

Hierarchy descriptors. One of the most intuitive properties to describe the complexity of a hierarchical segmentation is the number of levels it contains – the *depth of the hierarchy*, with

2 being the minimum possible depth (from the mother segment to all singletons) and $|V|$ the maximum (the mother segment losing a node at every level). Other structurally informative indicators are the number of communities and the number of splits, which indicate the amount of structural elements identified across the hierarchy and the amount of splits from which they originate. More precisely, a split is counted whenever a parent community at level i originates at least a new community at level $i+1$, in addition to the trivial one preserving the same nodes of the parent. Given that the depth of the hierarchy also depends on the music piece length, these three metrics are scaled with respect to the maximum values they can take in a given music graph. In this way, it is possible to compare tracks with different duration and metre.

Fragmentation metrics. These metrics describe key aspects of the decomposition trend of hierarchical segmentations, a peculiar property of music as suggested by Figure 7.1.

The *fragmentation imbalance* of a split is an indicator of how nodes distribute from a parent community C_l with $|C_l|$ nodes at level i to its children communities $C_{l,1},\ldots,C_{l,m}$ at level $i+1$. It ranges from 0, when $|C_{l,k}| = \frac{|C_l|}{m}\ \forall k \leq m$, to 1 for maximal imbalance – when $\exists\, C_{l,k}$ s.t. $|C_{l,k}| = |C_l| - m + 1$, with all the other new communities being singletons (Figure 7.3).

Because the fragmentation imbalance is computed from an individual split, obtaining a single metric for the whole hierarchy requires two steps of aggregation: first we aggregate the fragmentation imbalance of all the communities splitting between each couple of successive levels (*level aggregation*), then we aggregate across this hierarchy (*hierarchy aggregation*). We use the minimum, maximum and mean functions for level aggregation, and mean (M), standard deviation (SD), coefficient of variation (CV) and sample entropy (SampEn) for hierarchy aggregation. SD, CV and SampEn are used to study the dynamicity and predictability of the hierarchical fragmentation process. In particular, SampEn is a modification of approximate entropy that is independent from sequence length Richman and Moorman (2000). In statistical

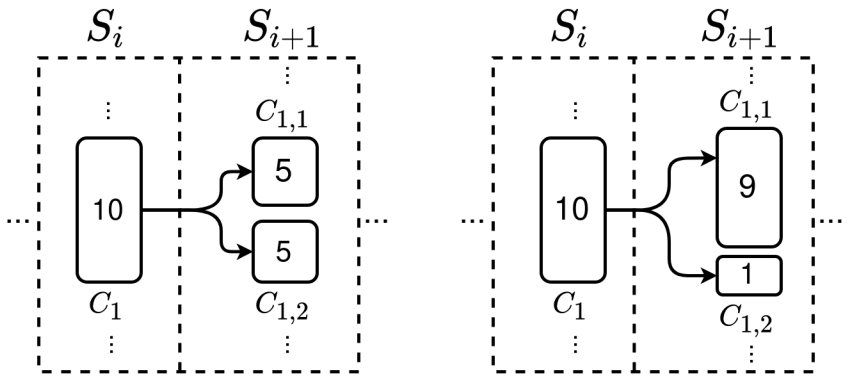


Figure 7.3: Examples of maximally balanced (*left*) and imbalanced (*right*) splits.

Group	Metric	Aggregation	Values	Description
Segmentation metrics	r_{min}	-	1	Smallest value of r to enclose all nodes in a single community.
	r_{max}	-	1	Smallest value of r to break the graph completely into singletons.
Hierarchy descriptors	hierarchy depth	-	1	Relative number of segmentation levels in the hierarchy.
	number of communities	-	1	Total relative number of detected communities.
	number of splits	-	1	Total relative number of community splits across the hierarchy.
Fragmentation metrics	singleton fragmentation	-	1	Propensity of communities to become singletons (single-node communities) towards the bottom of the hierarchy.
	no. of non-singletons communities	hierarchy	4	Relative number (proportion) of non-singleton communities per level, aggregated across the hierarchy (M, SD, CV, SampEn).
	size of non-singleton communities	hierarchy	4	Relative size of non-singleton communities per level, aggregated across the hierarchy (M, SD, CV, SampEn).
	fragmentation imbalance	level, hierarchy	12	Degree of distribution of nodes from parent to children communities, aggregated for each level (min, max, M) then across the hierarchy (M, SD, CV, SampEn).

Table 7.1: Taxonomy of the structural metrics and outline of their function in relation to the hierarchical segmentations.

signal processing, approximate entropy is used as a measure of irregularity and unpredictability of fluctuations of time-series data Pincus (1991). Sequences with several repetitive patterns receive small SampEn; less predictable (more complex) ones yield higher values.

To describe the degree of fragmentation of communities at a certain level, which indicates the persistence of nontrivial structural components, we consider the proportion of *non-singleton communities* together with their relative size. Given a segmentation level S_i the former one is obtained by counting the number of non-singleton communities and scaling it by the total number of communities in S_i. Similarly, the size of non-singletons communities – the number of nodes they contain, is scaled by $|V|$. As we obtain a time series for each metric – one for the number and one for the size of non-singleton communities per level, only hierarchy aggregation is required.

Finally, another metric – *singleton fragmentation* – describes how far in the hierarchy nodes tend to form singletons, indicating the pace at which the most atomic structural components – beats – tend to separate from larger structures. Given that the singleton fragmentation pertains to each $v \in V$, values are computed independently for each node and then averaged. More formally, assuming that a node v becomes a singleton in S_i (the i-th level in the structural hierarchy S), the fragmentation imbalance of v is simply defined as $\frac{i}{|S|}$, where $|S|$ denotes the hierarchy depth (the total number of segmentation levels in S). This metric ranges in the $[0, 1]$ interval. Values close to 0 indicate a slower and persistent fragmentation of the graph; if nodes tend to become singletons towards the end of the hierarchy, values would tend to 1.

7.2 Experimental evaluation

In this section, we describe our test framework, experimental procedures and results pertaining to the investigation of metrics to quantify the structural complexity of music pieces.

7.2.1 Music dataset

Our first step was to create a music dataset that included music with different levels of structural complexity. To achieve this, we included three types of subsets, which we think establishes a good test-bed: *human-composed* music (high complexity; "real" music), *computer-generated* music (which we expect will have intermediate complexity) and *random* music (minimal complexity). In our view, each of these subsets can be associated with a different level of structural complexity, which allows to investigate whether our metrics permit discriminating between these broad complexity levels.

Human-composed (real) music

This subset includes a selection of "real" music, i.e., music written by human composers, which includes a partition of the Pianomidi (Bernd, 1996) and SALAMI (J. B. L. Smith et al., 2011) datasets. The first, is a well-known dataset of classical music for piano (in symbolic format) spanning from the baroque era to the late Romantic and impressionist periods. The second, is a dataset used for audio-based MSA, providing live performances of pop/rock/blues music together with their structural annotations. We expect real music[1] to exhibit the highest degree of structural complexity, with short-term (e.g., motifs), mid-term (e.g., phrases) and long-term (e.g., sections) structural elements emerging from the compositions.

Computer-generated music

This subset includes music generated by three state-of-the-art machine learning models: the *Basic RNN*, the *Lookback RNN* and the *Attention RNN* (Elliot, 2016). These three models are particularly interesting because they have different levels of ability to produce musical content with long-term structures. Furthermore, all these models have a comparable number of learnable parameters, use the same encodings of symbolic music, and were trained on the same music corpus using similar strategies and optimisation methods. This ensures that the musical properties of the generated compositions – and in particular, their increased level of structural complexity, can be attributed to the architectural design of these models rather than being the

[1] The terms *real* and *human-composed* music are used interchangeably.

result of other factors that we would not be able to trace. Therefore, these models provide a controlled testbed for our experiments.

The *Basic RNN* is a vanilla LSTM recurrent neural network Hochreiter and Schmidhuber (1997) trained as a music model (one-step ahead prediction) on symbolic music sequences. This architecture is representative of several works in the literature, from the first attempts at music modelling with LSTMs (Eck & Schmidhuber, 2002a) to more recent architectures, such as the FolkRNN (B. Sturm, Santos, & Korshunova, 2015).

The *Lookback RNN* is an extension of the Basic RNN that introduces time-delayed connections (c.f. Section 3.1.2), and requires several additional inputs at each time step – a rich conditioning signal. More precisely, in addition to the previous musical event/token and assuming that all pieces have time signature of 4/4, a Lookback RNN receives the following information: (i) the specific events from one and two measures ago; (ii) whether the last token was repeating the event from one or two bars before it; (iii) two labels denoting whether the network has to repeat the event from one or two measures ago, respectively; (iv) the current relative position within the measure in terms of quarters. These architectural changes aim to facilitate the model to learn structural regularities from sequences by providing prior knowledge of metrical structure to the network Johnson (2017).

Finally, the *Attention RNN* is another architecture that expands the memory capacity of the LSTM by means of an attention mechanism (c.f. Section 3.5.2), which enables the network to contextually access the generated output sequence up to a certain number of elements. This frees the network from having to store musical content in the LSTM cell's state. Attention mechanisms have become staple in modern architectures for music generation (C.-Z. A. Huang et al., 2018), because they are more effective at modelling long-term dependencies in sequential data – a desideratum for the generation of music with increased levels of structural complexity (de Berardinis, Barrett, et al., 2020). Although generated music still does not posses a clear structural identity (Collins, 2009), autoregressive models tend to produce music exhibiting repetition and variation only at a local level (Boulanger-Lewandowski et al., 2012). Nevertheless, these subsets were specifically chosen due to the increasing level of structural complexity their music is expected to exhibit according to Elliot (2016).

(Quasi-)Random music

The last subset contains music artificially generated or manipulated in a way to compromise most of its structural integrity. First, we include a group of pieces generated with a Basic RNN (the same model outlined in Section 7.2.1) after a single epoch of training. Second, we used a scrambling method (Fedorenko, McDermott, Norman-Haignere, & Kanwisher, 2012) that randomly shuffles beat-aggregated feature vectors to destroy any structural relationship at the

measure	LA	HA	random-net	random-sa	random-pm	basic-rnn	lookback-rnn	attention-rnn	salami	pianomidi
r_{min}			-0.12 ± 0.04	-0.05 ± 0.03	-0.08 ± 0.05	-0.74 ± 0.35	-0.68 ± 0.3	-1.04 ± 0.24	-1.09 ± 0.16	$\mathbf{-1.11 \pm 0.11}$
r_{max}			4.2 ± 0.46	4.09 ± 0.22	4.1 ± 0.28	11.26 ± 10.03	11.17 ± 9.6	20.46 ± 14.32	$\mathbf{21.73 \pm 12.83}$	19.76 ± 9.0
hierarchy depth		-	0.08 ± 0.01	0.06 ± 0.02	0.07 ± 0.03	0.18 ± 0.08	0.16 ± 0.06	0.23 ± 0.07	0.25 ± 0.05	$\mathbf{0.28 \pm 0.05}$
number of splits		-	6.43 ± 1.07	5.21 ± 1.4	5.92 ± 1.28	9.27 ± 2.14	$\mathbf{9.83 \pm 1.5}$	8.18 ± 2.57	6.47 ± 2.32	6.53 ± 1.95
number of communities		-	$\mathbf{0.97 \pm 0.01}$	$\mathbf{0.97 \pm 0.0}$	$\mathbf{0.97 \pm 0.0}$	0.91 ± 0.1	0.92 ± 0.11	0.79 ± 0.17	0.72 ± 0.13	0.72 ± 0.11
singleton fragmentation		-	$\mathbf{0.74 \pm 0.0}$	$\mathbf{0.74 \pm 0.0}$	$\mathbf{0.74 \pm 0.0}$	0.7 ± 0.05	0.71 ± 0.04	0.66 ± 0.05	0.62 ± 0.04	0.62 ± 0.03
ns communities (number)		M	0.94 ± 0.03	$\mathbf{0.97 \pm 0.02}$	0.96 ± 0.03	0.68 ± 0.18	0.68 ± 0.13	0.54 ± 0.14	0.45 ± 0.13	0.45 ± 0.1
		SampEn	0.14 ± 0.02	$\mathbf{0.17 \pm 0.03}$	0.15 ± 0.02	0.11 ± 0.03	0.11 ± 0.02	0.09 ± 0.03	0.09 ± 0.02	0.09 ± 0.03
		SD	0.14 ± 0.07	0.07 ± 0.04	0.09 ± 0.05	0.35 ± 0.08	0.37 ± 0.06	$\mathbf{0.4 \pm 0.03}$	0.38 ± 0.04	$\mathbf{0.4 \pm 0.03}$
		CV	0.15 ± 0.08	0.07 ± 0.04	0.1 ± 0.06	0.59 ± 0.3	0.58 ± 0.21	0.79 ± 0.22	$\mathbf{0.92 \pm 0.22}$	$\mathbf{0.92 \pm 0.19}$
ns communities (size)	-	M	0.96 ± 0.03	$\mathbf{0.98 \pm 0.02}$	0.97 ± 0.02	0.76 ± 0.11	0.75 ± 0.09	0.69 ± 0.08	0.67 ± 0.06	0.66 ± 0.05
		SampEn	0.14 ± 0.02	$\mathbf{0.16 \pm 0.03}$	0.15 ± 0.02	0.09 ± 0.02	0.09 ± 0.02	0.08 ± 0.02	0.08 ± 0.02	0.08 ± 0.02
		SD	0.11 ± 0.08	0.04 ± 0.04	0.06 ± 0.05	0.31 ± 0.07	$\mathbf{0.34 \pm 0.06}$	$\mathbf{0.34 \pm 0.05}$	0.33 ± 0.03	0.33 ± 0.03
		CV	0.12 ± 0.09	0.04 ± 0.04	0.06 ± 0.06	0.43 ± 0.14	0.47 ± 0.11	0.49 ± 0.1	0.49 ± 0.07	$\mathbf{0.5 \pm 0.07}$
fragmentation imbalance	mean	M	0.13 ± 0.02	0.12 ± 0.02	0.12 ± 0.02	0.25 ± 0.1	0.22 ± 0.08	0.31 ± 0.08	$\mathbf{0.38 \pm 0.07}$	0.37 ± 0.06
		SampEn	0.56 ± 0.27	0.74 ± 0.38	0.67 ± 0.37	0.9 ± 0.5	0.82 ± 0.43	1.25 ± 0.53	$\mathbf{1.63 \pm 0.39}$	1.61 ± 0.39
		SD	0.17 ± 0.03	0.16 ± 0.03	0.17 ± 0.03	0.27 ± 0.07	0.26 ± 0.06	0.32 ± 0.05	$\mathbf{0.34 \pm 0.04}$	$\mathbf{0.34 \pm 0.03}$
		CV	1.38 ± 0.14	$\mathbf{1.41 \pm 0.14}$	$\mathbf{1.41 \pm 0.16}$	1.15 ± 0.2	1.22 ± 0.18	1.05 ± 0.16	0.9 ± 0.11	0.94 ± 0.11
	min	M	0.07 ± 0.03	0.06 ± 0.03	0.07 ± 0.03	0.14 ± 0.09	0.12 ± 0.07	0.2 ± 0.08	$\mathbf{0.26 \pm 0.08}$	0.25 ± 0.07
		SampEn	0.39 ± 0.15	0.48 ± 0.2	0.44 ± 0.19	0.38 ± 0.24	0.32 ± 0.18	0.53 ± 0.25	$\mathbf{0.74 \pm 0.27}$	0.71 ± 0.24
		SD	0.16 ± 0.04	0.15 ± 0.05	0.16 ± 0.05	0.26 ± 0.09	0.25 ± 0.08	0.32 ± 0.07	$\mathbf{0.36 \pm 0.05}$	$\mathbf{0.36 \pm 0.05}$
		CV	2.43 ± 0.47	$\mathbf{2.47 \pm 0.49}$	2.41 ± 0.4	2.28 ± 0.61	2.44 ± 0.58	1.85 ± 0.52	1.51 ± 0.32	1.5 ± 0.26
	max	M	0.22 ± 0.04	0.21 ± 0.04	0.22 ± 0.04	0.44 ± 0.13	0.41 ± 0.1	0.49 ± 0.1	$\mathbf{0.57 \pm 0.08}$	0.53 ± 0.08
		SampEn	0.54 ± 0.23	0.71 ± 0.37	0.66 ± 0.34	0.86 ± 0.39	0.87 ± 0.37	1.1 ± 0.34	$\mathbf{1.35 \pm 0.27}$	1.31 ± 0.24
		SD	0.25 ± 0.03	0.23 ± 0.04	0.25 ± 0.04	0.37 ± 0.04	0.37 ± 0.04	0.4 ± 0.03	$\mathbf{0.4 \pm 0.02}$	$\mathbf{0.4 \pm 0.02}$
		CV	$\mathbf{1.14 \pm 0.14}$	$\mathbf{1.14 \pm 0.12}$	$\mathbf{1.14 \pm 0.14}$	0.9 ± 0.18	0.94 ± 0.16	0.85 ± 0.15	0.72 ± 0.11	0.77 ± 0.11

Table 7.2: Overview of the structural measures computed on the dataset – mean and standard deviation are reported for each music subset, with the maximum values per-measure in bold. LA and HA denote level and hierarchy aggregation respectively.

beat level on all tracks of the SALAMI and Pianomidi datasets. We expect that music pieces from both these groups will have minimal structure.

Data overview

All sets comprise music pieces with duration of 180 ± 20 seconds – a duration we find suitable for the identification of structures spanning from short- to long-term scales. To obtain audio tracks, the MIDI files in our collection (all the generated music and the Pianomidi selection) are synthesised using `FluidSynth` and the freely available `FR3` General-MIDI soundfont[2]. The diversity of musical material (live performances, synthesised symbolic music) and the inclusion of different genres (classical, pop, rock, etc.) provides a challenging experimental setup to test the robustness and the generalisation of the structural metrics.

7.2.2 Methodology and results

Structural complexity metrics and summaries

Following the procedure detailed in Section 7.1.2, we computed all metrics for the music tracks in our dataset. To produce hierarchical segmentations, the coefficient of filtration of DMSCOM was set to its default value ($\lambda = 4$). Each metric was then grouped according to the music

[2] https://www.fluidsynth.org/

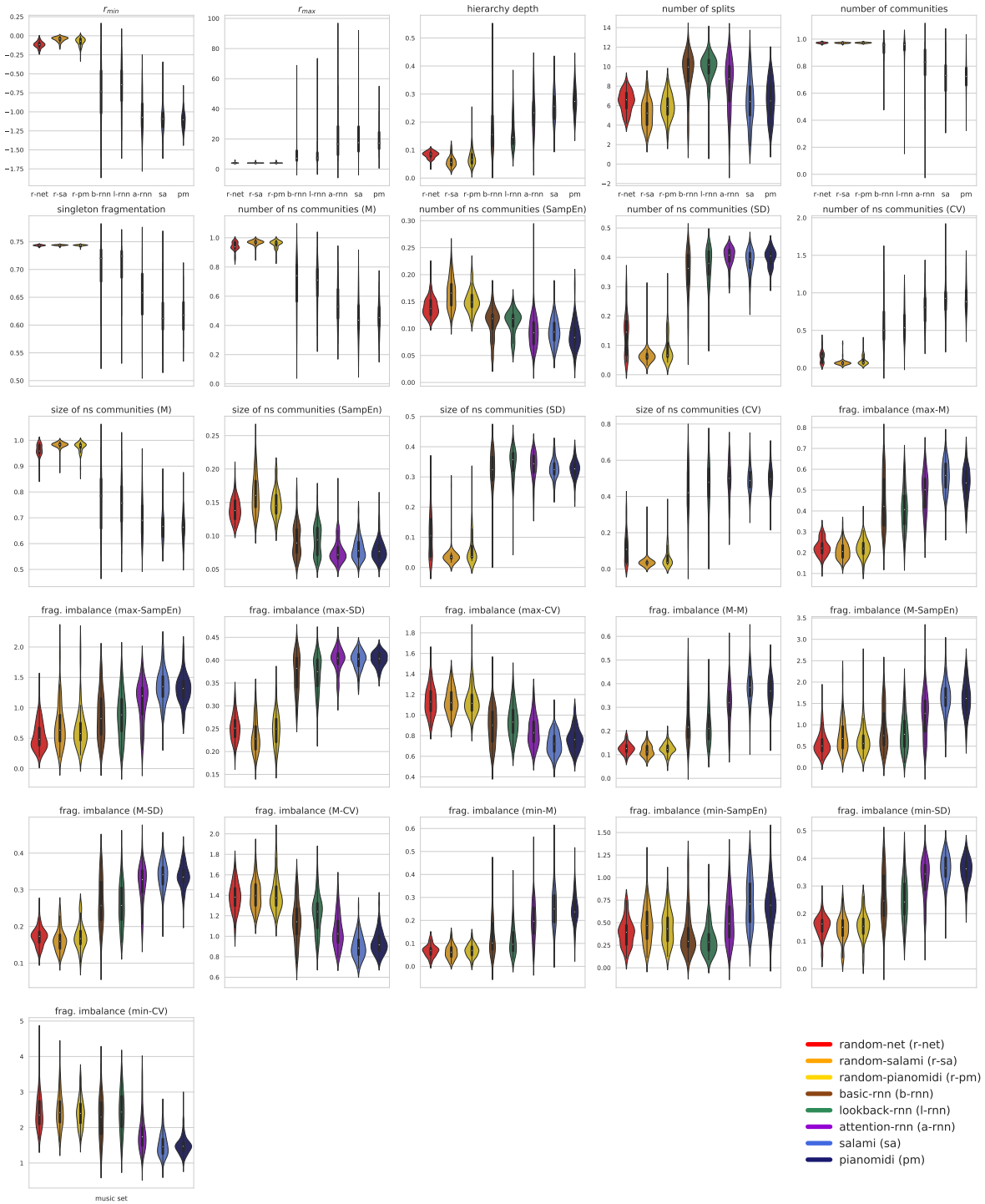


Figure 7.4: Overview of the structural metrics divided by each music subset in our collection – *random music* generated from an untrained LSTM (r-net) or by beat-shuffling on SALAMI (r-sa) and pianomidi (r-pm); and *computer-generated* music from a basic (b-rnn), lookback (l-rnn) and attention (a-rnn) LSTM; *real classical music* for piano selected from the SALAMI (sa) and pianomidi (pm) datasets. For those metrics requiring *hierarchy aggregation* only, the name of the functional is reported in brackets; when *level aggregation* needs to be applied before the former, both functionals are reported.

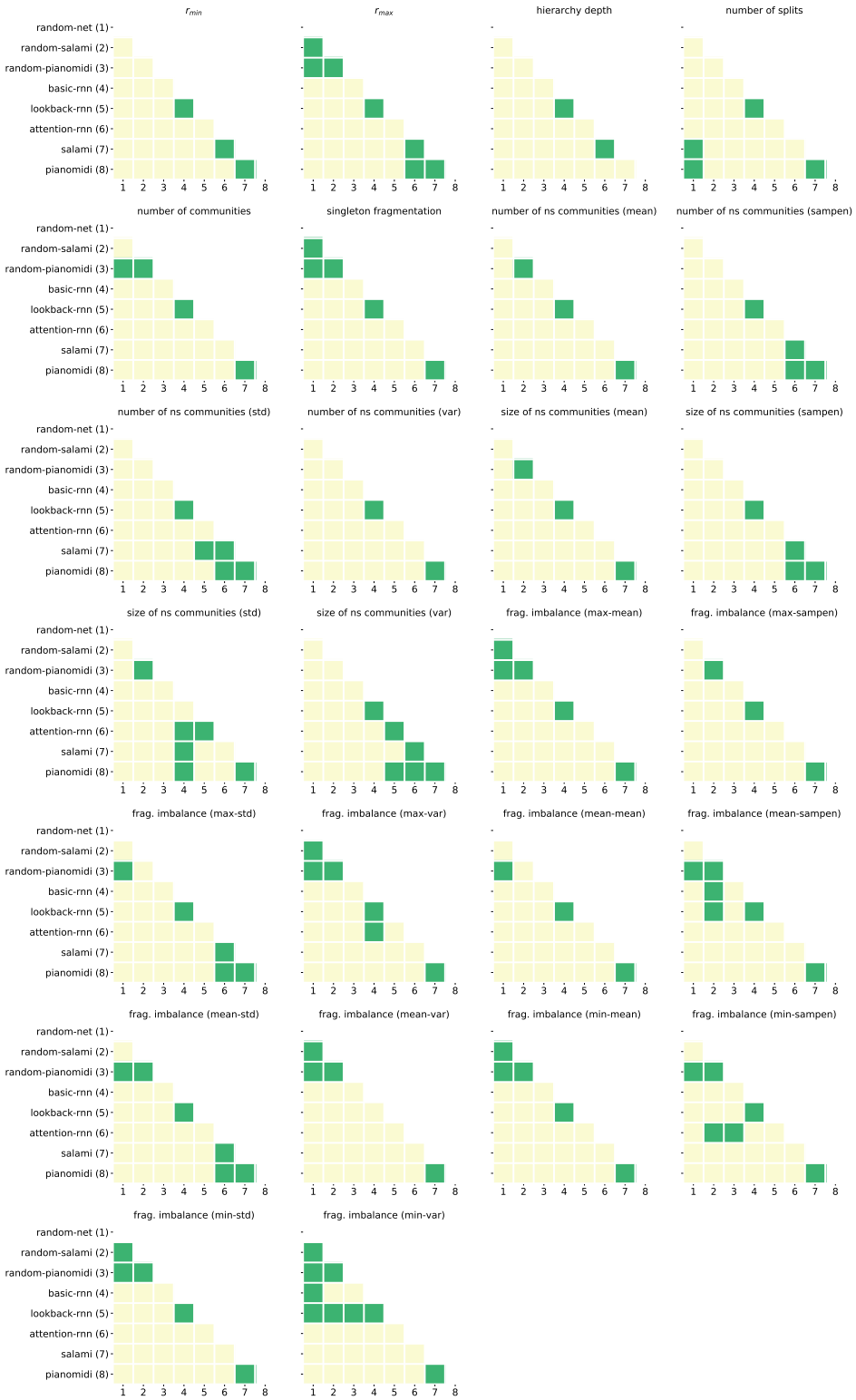


Figure 7.5: Pairwise statistical analysis of the music subsets for each structural metric, with yellow denoting statistical difference ($p < 0.05$) and green otherwise.

selection each track belongs to (e.g. Pianomidi), thereby enabling the analysis and the comparison of the distributions of these groups. This is illustrated in Figure 7.4 and is accompanied by Table 7.2, which reports the mean and the standard deviation of each metric per music selection. To detect statistically significant differences between the music selections, pairwise Kolmogorov-Smirnov tests were performed for all metrics (Bonferroni corrections were applied to control for family-wise error rate of multiple comparisons). This in-depth statistical analysis is reported in Figure 7.5.

From the results it was found that real music, compared to the other subsets, is harder to segment, as it is associated with deeper structural hierarchies following a more complex decomposition trend. The values of the resistance parameter r indicate that real music requires "more energy" to enclose all nodes within a single community (r_{min}), as well as to fully decompose networks into singletons (r_{max}). This is particularly evident between real and random music, with generated music only approaching the human-composed class with the *Attention RNN* – a pattern we found for several other metrics.

A similar trend was observed for the hierarchy descriptors. Hierarchies obtained from the segmentation of human-composed music are the deepest in terms of segmentation levels, with the fewest relative number of communities resulting from a reduced number of splits. In contrast, random music is segmented in shallow hierarchies with the highest number of estimated communities, although still originating from a few splits. Generated music, instead, sits in between the former groups for hierarchy depth and number of communities, stemming from the largest number of splits. Therefore, the relative number of splits is a structural property shared between real and random music, and allows to distinguish these groups from generated music.

Regarding the decomposition trend of music, a level-to-level analysis of non-singleton communities and their fragmentation across the hierarchies revealed interesting insights. For real music, the relative number and the size of non-singleton communities per level are the least complex to predict, with the highest standard deviation and coefficient of variation. Instead, the trend of non-singleton communities in random music is the most complex process, with the lowest amount of variation. The decomposition trend of human-composed music thus follows some regularity, which is particularly evident for the relative size of non-singleton communities.

The choice of level aggregation (LA) function did not influence the analysis of the fragmentation imbalance, as both *min*, *max* and *mean* provided similar insights. From this analysis, we found that real music produces the most imbalanced splits (0.38 and 0.37 average fragmentation imbalance with mean LA for SALAMI and Pianomidi respectively) compared to the other groups. The higher fragmentation imbalance of human-composed music, together with its reduced number of non-singleton communities per level, indicates a *leaky segmentation* behaviour. This means that the hierarchical segmentations of real music tend to be inflated by the

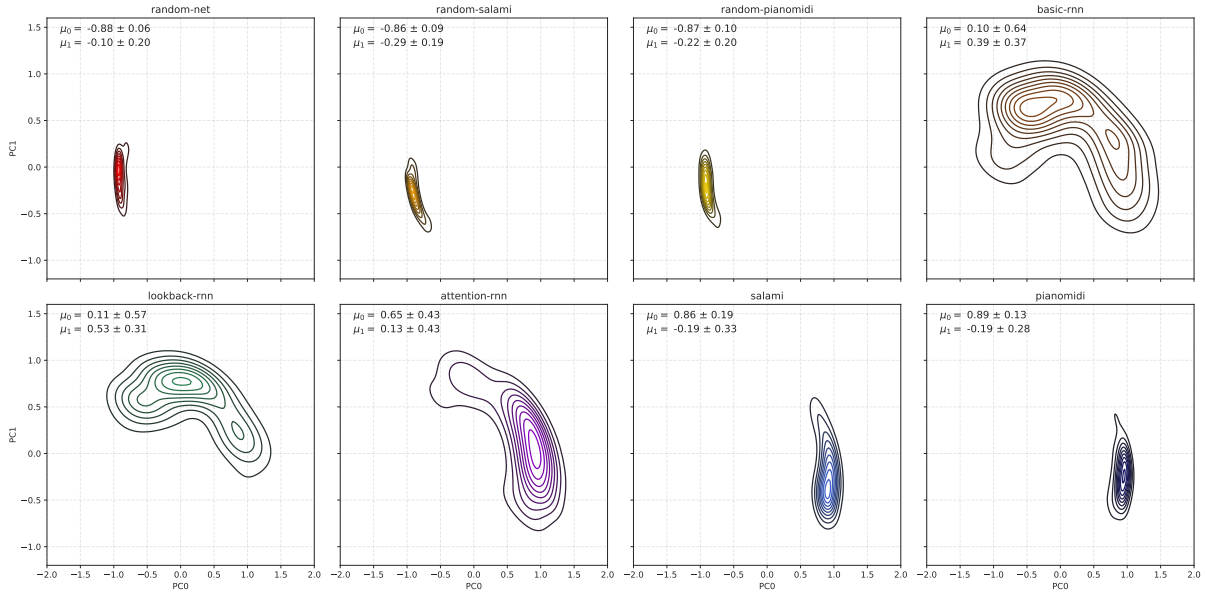


Figure 7.6: Illustration of the distributions of the structural summaries for each group after bivariate kernel density estimation. The mean and the standard deviation for each dimension (PC0 and PC1) are reported in the top-left of each plot.

number of nodes separating from larger communities as singletons, which in turn contributes to increase the hierarchy depth. The singleton fragmentation metric provides further insights into the pace at which this leaky segmentation occurs across hierarchies. Given that human-composed music has the lowest singleton fragmentation, the full decomposition of structural segments into singletons does not occur right at the bottom of hierarchies – a behaviour which is more pronounced for random and generated music. Indeed, the "leaky segmentation" of real music is more gradual throughout the hierarchies, rather than happening mostly towards their bottom levels. Overall, all these observations already allow to characterise each music group.

Structural complexity of different music subsets

As can be inferred from the analysis above, there is redundancy between the various metrics, which can indicate the existence of latent variables. This was confirmed via a correlation analysis, which revealed strong and significant correlation between more than 80% of the metrics. To identify potential latent variables, we employed principal component analysis (PCA) on the whole set of metrics after discarding the segmentation metrics. This ensures comparability of the latent variables independently of the MSA procedure used to produce hierarchical segmentations (r-values are DMSCOM-specific). The first two principal components explained 83% of the variance in the structural metrics. Hereinafter, we will focus on the first two principal components and refer to them as *structural summaries* – a compact descriptor of the structural

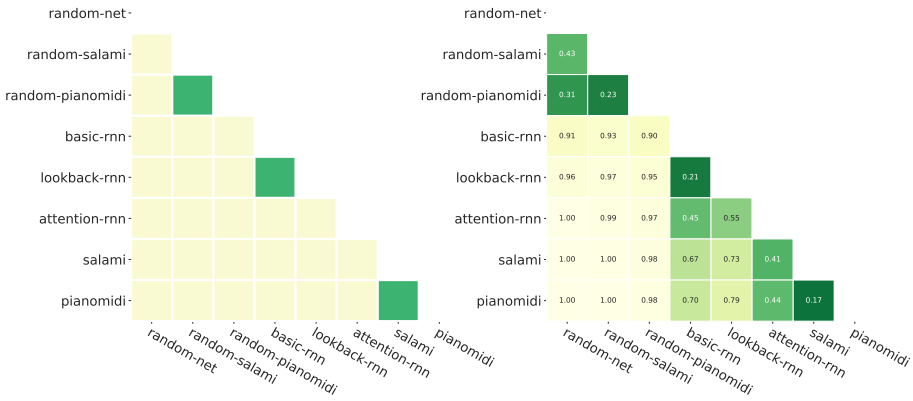


Figure 7.7: Statistical analysis of the structural summaries: pairwise comparison of the groups (*left*), with yellow denoting statistical difference and green otherwise; Kolmogorov-Smirnov scores (*right*) as a distance function between groups, illustrated with a colour-map ranging from green (maximum score, minimum distance) to yellow (minimum score, maximum distance).

properties captured by the original metrics.

As latent variables were identified, we analysed the distributions of the structural summaries (denoted as PC0 and PC1) of the various music subsets. These are plotted in Figure 7.6.

To detect differences between the distributions of each selection, we computed a series of Kruskal-Wallis H-tests and found that they differ significantly for both summaries ($\chi^2 = 630.85$ and $\chi^2 = 342.89$, $p < 0.0001$; for PC0 and PC1, respectively). The results of the pairwise comparisons conducted jointly for the structural summaries using bivariate Kolmogorov-Smirnov (KS) tests (after Bonferroni corrections) are shown in Figure 7.7. Due to the large number of comparisons, the results are reported as a *heatmap* which highlights statistically significantly different ($p < 0.05$) subsets in yellow and non-significant in green (similar distributions).

The pairwise analysis revealed five distinct clusters of structural complexity: (i) *random-salami* and *random-pianomidi* (a cluster that is identified as *randomised-human*); (ii) *random-net* on its own; (iii) *Basic RNN* and *Lookback RNN* (hereinafter, *simple RNN*); (iv) *Attention RNN* alone; and finally (v) "real" music (*SALAMI* and *Pianomidi*).

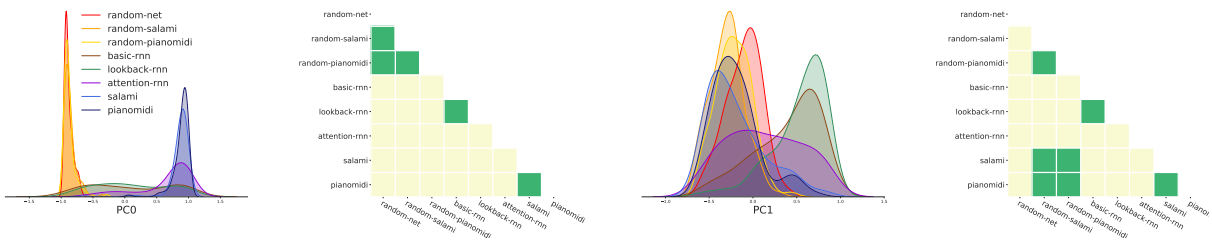


Figure 7.8: Illustration of the probability densities of the structural summaries (PC0 and PC1, considered independently) on each group. These are accompanied by the pairwise statistical analysis of the groups for each dimension, as done in Figure 7.7 for the bi-variate case.

A visual inspection of Figure 7.6 also suggested a specialisation of each structural summary: PC0 seems to distinguish between random, generated and real music (with low values associated to random music, and the highest for real compositions), and PC1 summarises structural properties which are common between random and real music (low values), but not for generated music (higher values). To evaluate these observations, pairwise comparisons were also conducted independently for each structural summary, using univariate KS tests (Figure 7.8).

As expected, PC0 differentiates between random and real music (no further subdivisions were found in each subset), whereas generated music still resulted in the *simple* and *attention RNN* clusters. Similarly to the bivariate case, the analysis of PC1 identified two distinct clusters for random and generated music, respectively. However, the distributions of PC1 associated to real music are not statistically different to those of their randomised counterparts (a single cluster encompassing both real and structurally-scrambled music), thus corroborating the hypothesis of structural properties shared by these groups.

In sum, the structural summaries allow discriminating between the different music subsets in our dataset, and can unveil further subdivisions in each of these groups. These subdivisions are retained structurally meaningful, as they confirm that scrambled music still preserves some degree of structure (as the perturbation is operated at the beat level), and the importance of attention mechanisms for AMC models; however, the architectural changes of the *Lookback RNN* did not significantly contribute to the structural complexity of the generated music compared to a vanilla LSTM model. Overall, the structural summaries provide a formal and automatic way to inspect where a given music piece or collection sits within the structural complexity plane.

7.3 Structural summaries: a framework for measuring music structural complexity

The analysis of the structural summaries on our dataset provides a compact and effective framework to evaluate the structural complexity of AMC models outputs. One possibility is to compare a corpus of generated pieces with the five reference classes of structural complexity we described in the previous section. Similarly to our previous analysis, Kolmogorov-Smirnov (KS) tests could then be used to compute the pair-wise differences between the bivariate distribution of the structural summaries extracted from the given collection and those of the reference classes. Furthermore, the resulting KS scores, ranging in $[0, 1]$, can then be interpreted as a dissimilarity measure between the generated corpus and each of these classes (Figure 7.7, *right*).

If the intention is to evaluate a single track, the Mahalonobis distance between its structural summaries (a data point) and the distribution of each reference class could also be computed.

	random-net	randomised-human	simple RNN	attention RNN	human
Original	28.68	22.42	2.08	0.90	**0.52**
Simplified	18.26	13.45	3.13	**3.00**	4.63
Randomised	1.22	**0.44**	3.67	5.63	11.65

Table 7.3: Mahalonobis distance of the structural summaries extracted from each version of Vivaldi's La Caccia w.r.t. the reference complexity groups. The distance of the closest reference class is highlighted in bold for each track.

An example of this approach is shown in Table 7.3 for Vivaldi's "La Caccia" (Autunno part III) – a classical music piece for orchestra from the Baroque period. In addition to the original orchestral version, we included: a structurally simplified version of the former piece, that is used for educational purposes (recorder practise in secondary school); as well as a randomised version of it, following the same scrambling procedure outlined in Section 7.2.1. As shown, both the original and the randomised versions received the smallest distance to their expected classes - *human* (0.52) and *randomised-human* (0.44), respectively. The simplified version, instead, has structural properties closer to those of generated music, and, in this particular case, to the *Attention RNN* outputs. These results are thus in line with the consideration that the structural simplification of the educational track was artificially operated to make it easier for novice students to analyse and play the piece on the recorder. Although the Mahalonobis distance of the structural summaries of the simplified version from their closest distributions – 3.13 from the *Simple RNN* and 3.00 from the *Attention RNN*, is not as low as those of the original and random versions, there is still reasonable margin to the other reference complexity classes.

From a statistical perspective, the use of our framework would be more reliable if distributions are to be compared, rather than individual tracks. This approach would also align to the expected use case for automatic evaluation. Experimenters would generate a reasonable number of tracks from their music generation system, extract a number of metrics to quantify specific musical properties of the compositions, along with their structural summaries. The latter would then be compared to the reference complexity classes for structural evaluation.

7.4 Chapter summary

In this chapter, we addressed the automatic analysis of structural complexity of music – an open problem in the field of computational music analysis which is currently jeopardising the systematic evaluation and the comparison of music generation systems. Our approach builds upon computational methods for hierarchical music structure analysis (MSA), capable of unveiling the nested organisation of music from long and articulated musical ideas (e.g. sections) to progressively shorter and simpler structural components (e.g. motifs). Given a music track, a

structural segmentation is first estimated as a hierarchical object using a state-of-the-art method for hierarchical MSA. This is followed by the extraction of a set of metrics to formally describe these hierarchies and the decomposition of music structures therein estimated.

To test the ability of our metrics to characterise structural properties of music, we computed them on a dataset including random, real and computer-generated music – groups which we expect to be associated with different degrees of structural complexity. After analysing their distribution on each group, we found that not only our metrics permit to discriminate between them, but further non-trivial subdivisions can also be identified according to the structural properties of the compositions. Our results thus revealed how these hierarchies differ as mathematical objects, and demonstrated the effectiveness of our metrics as structural descriptors.

We also showed how these metrics, together with their statistical analysis on the dataset, can provide a compact framework for automatically evaluating the structural complexity of a given collection of music or individual tracks. To the best of our knowledge, our method is the first to achieve this and comes with the following strengths: (i) our metrics exclusively capture structural aspects of music, due to the MSA step; (ii) it is fully unsupervised; (iii) there is no subjectivity involved in the definition of structural complexity; (iv) it can be adapted to deal with arbitrary signals. In addition, as our method takes music recordings as input, the resulting framework can be used to evaluate both audio-based and symbolic music generation systems, although a sonification step of compositions is needed in the latter case.

Overall, building upon the previous chapter, this work demonstrated that structurally informative descriptors can be effectively derived from the hierarchical segmentations of music, and contributed to the automatic evaluation of structural complexity in computer-generated music – thereby addressing RQ 2, in full. The structural complexity metrics can be easily extracted from music recordings produced by any computational method and compared to their reference distributions corresponding to quasi-random (little or no structure) and real music (structure at all levels). This provides a continuum, ranging from quasi-random to real music, where computer-generated music can be effectively evaluated and compared for structural complexity.

The next chapter will look at utilising these metrics as complementary audio features for music emotion recognition, and also, how other alternative structural properties of music (instrumentation) can be leveraged to increase the interpretability of that task.

Chapter 8

An investigation of structure in music emotion recognition

The final contribution chapter addresses the last research question (RQ 3) by investigating the use of music structural properties in the domain of music emotion recognition (MER). This is done by pursuing two parallel research directions: (i) by leveraging the structural complexity metrics introduced in the previous chapter (c.f. Chapter 7) – where structure refers to "music form" – in accordance with the other research questions; and (ii) by leveraging another structural property of music, i.e., instrumentation, to increase the interpretability of MER models.

More precisely, the first quest focuses on the potential role that structural complexity of music can play in inducing emotions on the listeners. To that aim, Section 8.1 tests the use of our structural complexity metrics as audio features for MER. This is done via a regression analysis involving two different experiments: a feature selection study to understand whether structural metrics can provide additive information when coupled with state-of-the-art audio features for static MER; a randomisation experiment to test whether using the structural metrics alone – as the only input features for arousal-valence regression, can improve the random baselines.

In contrast, Section 8.2 focuses on instrumentation – another structural property of music that is not related to form, but still contributes to the rich and complex texture of the music signal. Through a comparative experiment including state of the art models for MER, we introduce a new architecture leveraging music source separation techniques and demonstrate that it can improve the performance and the interpretability of these methods. This last work was presented in (de Berardinis, Cangelosi, & Coutinho, 2020).

8.1 Structural complexity in MER

The link between structure and emotional expression of music has been one of the promising yet unexplored research directions in music psychology. From the seminal work of Sloboda (1991), asking participants to locate specific musical passages that reliably evoked their emotional responses, a number of contributions addressing the structure-emotion synergy followed. Among these works, (Gomez & Danuser, 2007) explored the relationships between 11 structural properties of music and both self-reports of felt pleasantness and arousal and different physiological measures, such as respiration, skin conductance, and heart rate. The relationships between structural features and experienced emotions were thus found to correlate well with those known between musical structure and induced emotions. Their results suggested that "*the internal structure of the music played a primary role in the induction of the emotions in comparison to extra-musical factors*" (Gomez & Danuser, 2007). In (Livingstone et al., 2012), continuous measures of arousal and valence were compared across music that contained repetitions, variations, or contrasting segments – all central elements in music structure analysis, as outlined in Section 2.2. From this study, they found that arousal ratings differed the most for contrasting music, moderately for variations, and the least for repeating musical segments. Furthermore, Livingstone et al. (2012) trained a computational model to detect repeated music segments directly from listeners' emotional responses, and found that the model could detect the locations of phrase boundaries better than using other music/acoustic properties, such as performed tempo or physical intensity.

On a more general note, other works sought to determine which musical properties contribute to emotional expression. For instance, Eerola, Friberg, and Bresin (2013) designed a system that allows real-time changes to be made at playback to both structural (articulation and mode) and expressive cues (tempo, pitch, dynamics, brightness) of a musical piece. By manipulating such musical cues, the authors could determine whether they operate in additive or interactive fashion in relation to emotional expression.

In sum, there has been research across different fields around this link, but the influence of music structure on emotional expressions has not yet been proved computationally. More precisely, what is still not clear, is whether structural properties of music can provide additive information to the computational task of music emotion recognition.

8.1.1 Towards structural features

To initiate this novel direction, here we propose leveraging summative descriptors of structural complexity as complementary input features to train MER systems. Our goal is to investigate whether structural features can provide additive information for MER, so that they can be used

to increase the performance and the interpretability of the current methods.

As introduced in Section 2.3, MER models are usually trained either on previously extracted ad-hoc audio features, or low-level time-continuous acoustic descriptors derived from the Fourier transform (e.g. mel spectograms), or directly on the raw audio signals – thus operating in the waveform domain. Approaches based on the latter features make extensive use of deep neural networks, and can achieve state of the art performance thanks to their ability to learn complex transformations directly from the audio level. Nevertheless, the use of handcrafted and formally defined input features has clear benefits when it comes to the explainability and control of the resulting systems – both notorious drawbacks/limitations of methods built on neural networks. This last property is particularly important when the behaviour of a MER model needs to be interpretable in relation to its predictions, thus allowing to trace which musical feature/property might be triggering a certain emotional response to listeners. Explainability is indeed desirable whenever MER systems are used for music generation (Madhok, Goel, & Garg, 2018), personalised music recommendation (Bodarwé, Noack, & Jean-Jacques, 2015), and mood regulation (Coutinho, Alshukri, de Berardinis, & Dowrick, 2021).

With these premises, here we focus on the former case, as the use of predefined audio features also provides a natural testbed for the investigation of our structural complexity metrics as complementary features for MER. Given an audio track, both the structural complexity metrics discussed in Chapter 7, and a set of summative audio features for MER are extracted simultaneously from the recordings; the combination of both feature sets is then considered for static MER, where a model is trained to predict either a single emotion class or a $(\text{valence}, \text{arousal})$ couple for each track. Hence, this kind of setup cannot be used for time-continuous MER, as the structural complexity metrics would not be meaningful if computed at short time scales.

Examples of common audio features used to train MER systems include energy, spectral, cepstral and voicing related low-level descriptors, as well as measures of spectral harmonicity, psychoacoustic spectral sharpness, and harmonic-to-noise ratio (HNR). These are all part of the `ComPaRe` audio features (Schuller et al., 2013) – a reference feature set for emotion recognition from speech and music, which has been extensively used for the INTERSPEECH challenge. As these audio features are extracted at a very granular level in the first instance (usually corresponding to a short temporal window), a common procedure is to aggregate these frames at a coarser level (e.g. 500ms) using a series of functionals (mean, standard deviation, min, max, etc.). Given that the structural complexity metrics are computed at the song-level, following the structural segmentation of tracks (c.f. Section 7.1.2), audio features need to be aggregated at the song-level, thus resulting in a single vector – a setup that is commonly known as *summative*. Therefore, this provides a full set of summative audio features for static MER.

8.1.2 Experimental evaluation

To validate our structural complexity metrics as complementary audio features for MER, a series of static MER experiments were conducted, with regression models trained to predict valence and arousal. As the main objective is to demonstrate that these features can provide additive information for this task, the methodology is organised in two different experiments. First, the extended feature set – including both the structural complexity metrics and the audio features, was considered for a feature selection experiment. The goal of this experiment is to test whether a subset of the former is retained in the final selection, which is expected to collect only those that are deemed useful for the regression task. Furthermore, in a second experiment different MER regressors were trained solely on the structural complexity metrics to see if they can surpass their corresponding baselines trained on a randomised set of the structural features.

Dataset

For both experiments, we considered a subset of the *MediaEval Database for Emotional Analysis of Music* (DEAM) – a reference collection for music emotion recognition (Aljanaki, Yang, & Soleymani, 2017a). The database consists of an aggregation of three datasets from the "Emotion in Music" task at the MediaEval benchmarking campaign 2013-2015 (one dataset per year), and comprises royalty-free music to permit the distribution of the audio material. The whole corpus includes 1802 songs belonging to 14 musical styles – Soul, Blues, Electronic, Rock, Classical, Hip-Hop, International, Folk, Jazz, Country, World, Rap, Raggae and Pop (with a maximum of five songs per artist). From those, 1744 are excerpts of 45 seconds from full songs taken from the Free Music Archive (`freemusicarchive.org`) and 58 are full length songs taken from the medleyDB collection (Koory & Medley, 1987) and Jamendo Music (`jamendo.com`).

All tracks were annotated in terms of the emotions that are expressed by the music in terms of *arousal* and *valence*, and the annotations available are both dynamic (i.e., on a moment by moment basis) and static (i.e., a single final judgement for the whole song). The dynamic annotations were collected using a web-interface on a scale from -10 to 10 (valence and arousal dimensions were annotated separately) whilst the music was being played. The static annotations were made on a 9-point rating scale – for both valence and arousal, after the dynamic annotations were completed. For more details please refer to (Aljanaki et al., 2017a).

Given that the structural complexity metrics are more meaningful when computed on full-length audio tracks, both experiments consider the static valence/arousal annotations and assume the availability of the original recordings. As DEAM provides 45s audio excerpts for most tracks, the Free Music Archive (FMA) was queried based on the metadata accompanying the database – including the title, artist and release of each track in the collection. Matching

procedures operating on these fields were then applied to retrieve the corresponding tracks from the FMA and download the audio recording covering the entire duration of these songs.

Due to the limited size of DEAM, preventing the use of a predefined training-validation-test split, both experiments nested cross-validation (CV) with 5 outer-folds and 5 inner-folds, respectively. In other words, the dataset was split in 5 distinct partitions, where each of them is considered as test set and the remaining form the development set (training and validation) at each round. To avoid biasing results on the choice of training and evaluation sets, another CV step is recursively applied on each development set (for each outer-fold). Hence, 25 different runs are performed in total (5 for each outer fold) and results are always aggregated across all test partitions or reported for each of them as separate experiments.

Feature selection experiment

To test the hypothesis that the structural complexity metrics can provide complementary information for music emotion recognition, when combined with conventional audio features, the first experiments aims at evaluating such interactions via a feature selection study.

More precisely, an extended feature set is obtained by extracting and concatenating state of the art audio features for MER with the structural complexity metrics introduced in Section 7.1.2. The resulting structurally-extended feature set, partitioned according to the nested CV procedure detailed before, is then considered for feature selection. The latter was performed with `Boruta` – an all-relevant feature selection method that attempts to find all features carrying information usable for prediction (Kursa, Rudnicki, et al., 2010). Notably, `Boruta` differs from traditional feature selection methods finding subset of features on which some model has a minimal prediction error; conversely, it finds all features that are either strongly or weakly relevant to the response variable (in our case, either valence or arousal). Therefore, this algorithm turns out to be ideal to test the following hypothesis: if one or more structural metrics are retained after feature selection, then it is possible to conclude that these features contributed – together with the selected audio features – to the prediction of valence/arousal; otherwise, the hypothesis is rejected in case all the structural metrics are discarded after feature selection, as the audio features would already provide all the relevant information for MER.

Concerning the audio features, in this experiment we use the novel Polyhymnia feature set from (Coutinho & Schuller, 2021), providing state of the art acoustic and musical features for emotion expression. This set complements `ComParE` (Schuller et al., 2013) with new features capturing specific properties related to loudness, timbre, rhythm, roughness, melody, and tonality. Hence, none of these features directly encodes information related to music form, which is instead one of the objectives of the structural complexity metrics.

To test the relevance of the structural features in different settings, two different setups

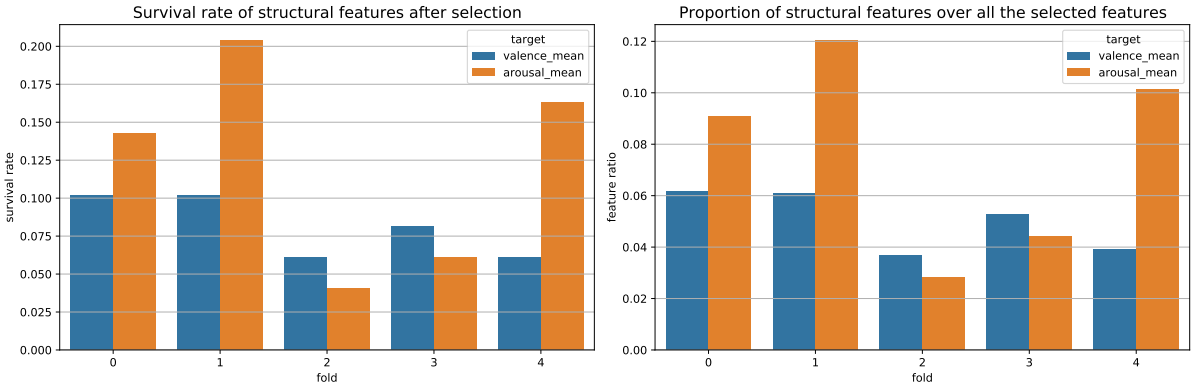


Figure 8.1: Hybrid setup: percentage of structural features retained after feature selection (*left*), and proportion of structural features within the selected feature set (*right*), which includes all audio features (i.e. the Polyhymnia features) for the remaining part. This is reported independently for *valence* and *arousal*, and with respect to each outer fold of the dataset.

are considered for this experiment: (i) *hybrid setup* – audio features (Polyhymnia feature set) are extracted from the 45s audio clips, whereas structural features are computed on full-length tracks; (ii) *all clips setup* – both feature sets are extracted from the 45s audio clips.

Results. The results of the feature selection experiment are illustrated in Figures 8.1 and 8.2 for the hybrid setup, and in Figures 8.3 and 8.4 for the all-clips setup. For both setups, the first figure shows the percentage of structural features that were retained after feature selection (left-hand side), and the proportion of structural features over all selected features (right-hand side). Each of these plots reports the outcomes of feature selection performed on each outer fold and with respect to either valence or arousal, thereby allowing to asses the relevance of features to predict each response variable separately. The second figure shows the name of the selected structural features together with the probability of finding them selected across the different folds. For example, a structural feature associated to 0.8 was found in 4 out of 5 outer folds.

As can be seen in Figure 8.1, when using full-length tracks for the extraction of the structural features and 45s-clips for the audio features (the *hybrid* setup), the average survival rate of structural features is 8.2% for valence and 10% for arousal. The proportion of structural features in the selected feature set is 5% for valence and 7.5% for arousal, on average; hence, the remaining ones in the selection are all audio features (the Polyhymnia features). In particular, the structural features that are more likely to be selected for valence (at least 0.6 selection probability) are the fragmentation ratio of communities (max aggregation) and the total number of splits (Figure 8.2). This last feature was also selected for arousal, in addition to some fragmentation imbalance metrics (mean/mean, mean/var, and max/mean aggregation).

When considering audio clips for the whole feature extraction pipeline (the *all-clips* setup), structural features turned out to be more useful for valence. In fact, the average survival rate

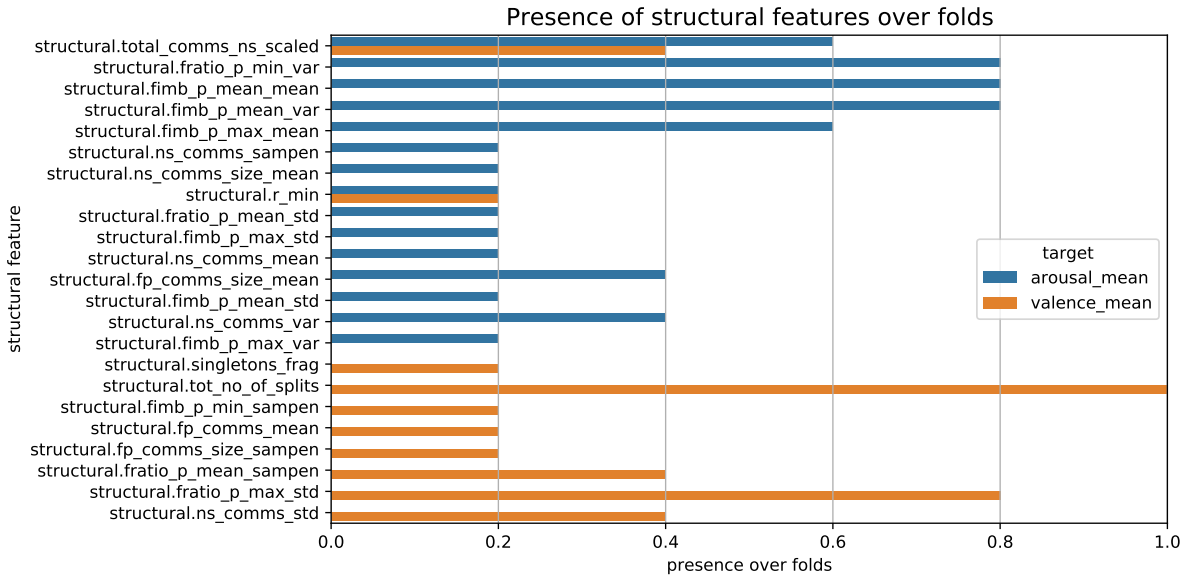


Figure 8.2: Hybrid setup: probability of finding a structural feature in an outer fold after feature selection (probability of selection), given the response variable to predict (valence or arousal).

of structural features after selection was 14% for valence and 8.5% for arousal (Figure 8.3). Within the selected feature sets, the proportion of structural features was 9% for valence and 5% for arousal, on average. Notably, also the most relevant structural features varied from the previous setup: singleton fragmentation, r_{min}, non-singleton communities (standard deviation aggregation), and fragmentation imbalance (sample entropy aggregation) were the most likely to be selected for arousal; for valence instead, these were fragmentation ratio (min/var aggregation), number of communities (sample entropy aggregation), r_{max} and maximum modularity. Overall, in the all-clips setup, 10 structural features were selected for arousal and 14 for valence (Figure 8.4), in contrast with the hybrid setup which counted 15 for arousal and 10 for valence.

Randomisation experiment

The previous experiment demonstrated that our structural metrics carry information usable for valence and arousal prediction, when complemented with state of the art audio features for MER. What remains to be proved, is whether this kind of additive information also emerges when structural features are considered independently, rather than relying on the interaction with the audio features. In other words, using solely the structural metrics as input features for emotion recognition should statistically improve the regression performance of some baselines.

For this purpose, a separate experiment was carried out to compare the emotion recognition performance of a pool of regression models under two different setups – that only differ on the input features these models receive for valence/arousal prediction. In the first setup, models

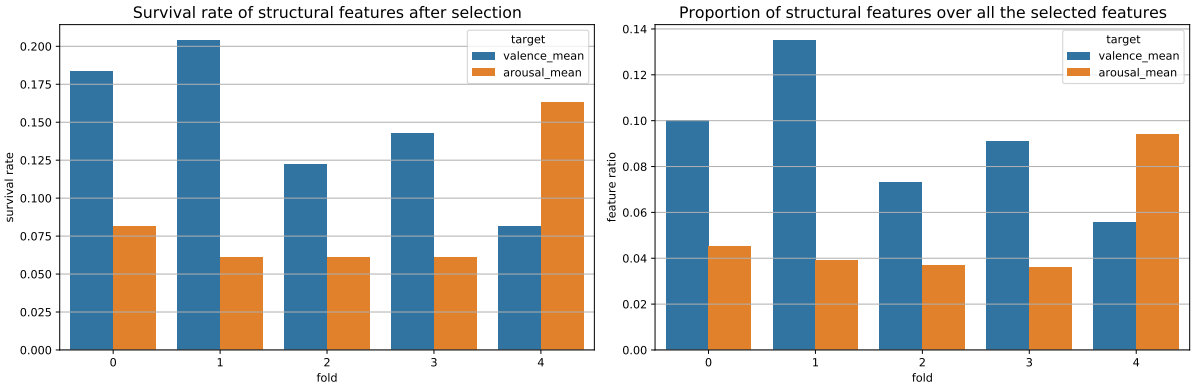


Figure 8.3: All-clips setup: percentage of selected structural features (*left*), and proportion of structural features within the selected feature set (*right*), as in explained in Figure 8.1.

receive all the structural features computed on full-length tracks (the same structural features used in the *hybrid* setup). In the second settings, the input features consist in a record-wise randomisation of the structural features where each of them was shuffled across all tracks (e.g. a track will swap the value of each structural feature with that of another track). For both setups, regression models are trained to predict valence and arousal from their respective feature sets, and evaluated on each test partition using the root mean squared error (RMSE) and the coefficient of determination (the R^2 score) between the models' predictions and the groundtruth annotations (for each target variable). The coefficient of determination is commonly used to complement the RMSE for the evaluation of regression models: when the R^2 score is maximum (equal to 1), the variability of the target data is fully captured by the model; conversely, a score equal to 0 corresponds to the case where a model predicts the expected value of the target.

The main hypothesis of this experiment can thus be reformulated as follows: any regression model trained on the structural features to predict valence and arousal will always outperform its counterpart trained on the shuffled features. Together with the previous experiment, the goal is indeed to demonstrate the independence between structural and traditional audio features, hence the ability of the former to provide additive and intrinsic information usable for MER.

In relation to the regression models, we tested a variety of methods including support vector machines for regression (SVR), kernel Ridge regression (KRR), and random forests (RF). For SVR and KRR, three versions based on different kernel functions were considered – using linear, Laplacian and Radial basis functions. In addition, single- and multi-target learning frameworks were tested for KRR and RF, which are denoted as `concat` and `all`, respectively. For each nested CV inner fold, the hyper-parameters of each regression model (e.g., the kernel coefficients, regularisation coefficient for the SVR, etc.) were optimised via grid search.

Results. The evaluation results of the regression models trained on the structural features –

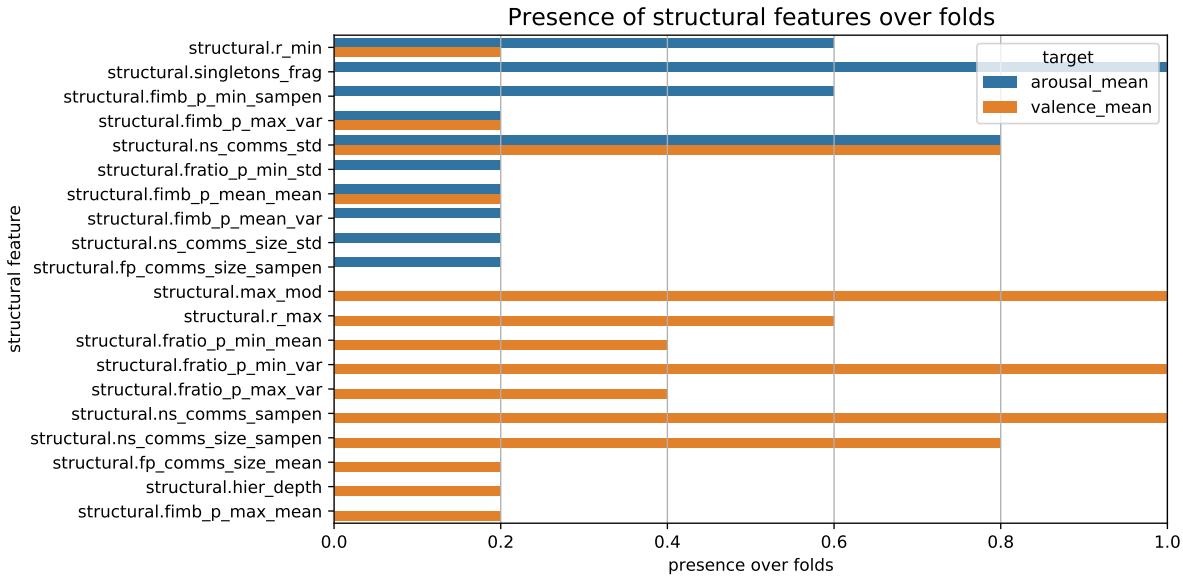


Figure 8.4: All-clips setup: (non-null) selection probabilities of structural features across outer folds, given the response variable to predict (valence or arousal).

with no perturbation – is reported in Table 8.1. This is accompanied by Figure 8.5, which quantifies the difference in performance between the best regression models trained on the original structural features (the same reported in Table 8.1), and those trained on the shuffled setup.

As can be noticed from Table 8.1, all regression models trained on the structural features outperform the baseline returning the expected value for both valence and arousal, which trivially achieves unit RMSE and zero coefficient of determination due to the standardisation of input features and target annotations. In line with previous work, valence is the most challenging dimension to predict for MER, as it emerges from the worse performance of all regression models (higher RMSE, lower R^2) when compared to arousal. The best regression model is the KRD with Laplacian kernel trained to predict both the target annotations (`krd_laplacian_all`), achieving RMSE of 0.956, 0.935 and R^2 of 0.086 and 0.126 for valence and arousal respectively.

When comparing these evaluation results with the corresponding metrics from the shuffled setup, by subtraction, a consistent gain in regression performance is observed for every model (c.f. Figure 8.5), with the trivial exception of the dummy model (which does not use any input features as it just outputs the expected value). To conclude, this last finding allows to accept the hypothesis formulated for this experiment, thus confirming that structural features are independently usable for music emotion recognition. In fact, if the structural features did not provide any usable information for MER, the regression performance of the first setup would have been similar, if not identical, to that of the second setup. In other words, shuffling the structural features would not have produced any change in terms of regression performance.

Regressor	RMSE valence	RMSE arousal	R^2 valence	R^2 arousal
dummy (expected value)	1	1	0	0
linear_regression	0.9832	0.9491	0.0333	0.0992
svr_linear	0.9694	0.9449	0.0603	0.1072
svr_rbf	0.9632	0.9483	0.0723	0.1008
svr_laplacian	0.9581	0.9430	0.0820	0.1107
krd_linear_all	0.9639	0.9446	0.0709	0.1077
krd_rbf_all	0.9625	0.9490	0.0737	0.0993
krd_rbf_concat	0.9629	0.9492	0.0728	0.0991
krd_laplacian_all	0.9560	0.9349	0.0860	0.1259
random_forest_all	0.9644	0.9432	0.0699	0.1104
random_forest_concat	0.9668	0.9450	0.0654	0.1070

Table 8.1: Test results of regression models trained exclusively on the structural measures. The first entry is only provided as a reference baseline that always returns the expected value for arousal and valence; hence it achieves unit RMSE (as both input features and target annotations are standardised to zero mean and unit standard deviation) and the lowest R^2 score.

8.2 Further structural dimensions: a look at instrumentation

Having demonstrated that the structural features provide additive and independent information for MER, the investigation can proceed to other structural properties of music that are not directly related to musical form. In particular, this section looks at the structural texture created by the superimposition of instruments, in relation to their function in music emotion recognition.

As different voices within a composition can have a distinct emotional impact (Huron, Anderson, & Shanahan, 2014), our work leverages state-of-the-art deep learning methods for music source separation (MSS) to reduce the complexity of MER when limited training data is available. The proposed architecture (EmoMucs) is based on combining source separation methods with a parallel block of source-specialised models trained independently, and subsequently aggregated with a fusion strategy (Section 8.2.1). To benchmark this idea, EmoMucs was evaluated on the popular music with emotional annotations (PMEmo) dataset (K. Zhang, Zhang, Li, Yang, & Sun, 2018), and its performance were compared to reference deep learning models for MER (Section 8.2.2). Experimental results demonstrated that EmoMucs outperforms the baselines for valence recognition, and achieves comparable performance for arousal, thus providing increased interpretability at no cost of explainability.

8.2.1 Emomucs: the music emotion multiplexer

Our approach is based on the observation that different musical sources in a composition can evoke distinct emotional responses from the listeners (Huron et al., 2014). Given a music piece,

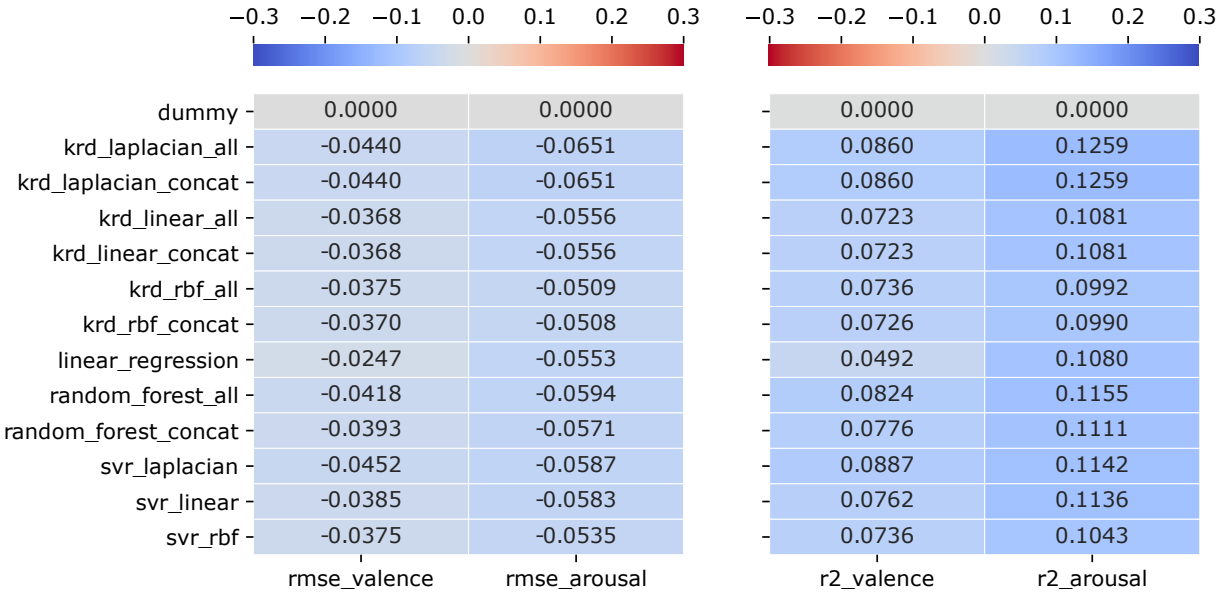


Figure 8.5: Difference in regression performance between the best models trained on the original structural features (first setup) and those trained on the shuffled features (second setup).

they can contribute differently to the overall induced emotion. For instance, the vocal lines of a track may be more informative to predict valence, whereas drums might have more impact on arousal. Nevertheless, the goal here is not to provide a general explanation of the emotional influence of musical parts, as this is often an individualistic property belonging to each track. Instead, a novel computational model for MER was proposed, leveraging the decomposition of the audio signal to the possible sources (e.g. vocals, drums) that can be detected from it.

By doing so, it would potentially be easier for the model to process the audio stream whilst searching for emotion-related patterns in every single source. The aggregation of the resulting source-specific models within a single architecture would also account for the possible inter-source relationships. This approach can thus be considered as a way to provide prior knowledge to a model in order to reduce the complexity of the learning task when limited data is available – a recurring issue for music emotion recognition.

Considering the technical challenges in MER, the design of a computational model based on music source separation has the potential to (i) improve the performance of the current solutions with the same amount of training data; (ii) provide a modular architecture which can be further adapted and fine-tuned with respect to each source-specific module, and (iii) quantify the contribution of each source to the final prediction for improved interpretability. The proposed model, EmoMucs, achieves this through a multiplexed framework for emotion recognition. The architecture of the model is illustrated in Figure 8.6, and its building blocks are explained in the following subsections.

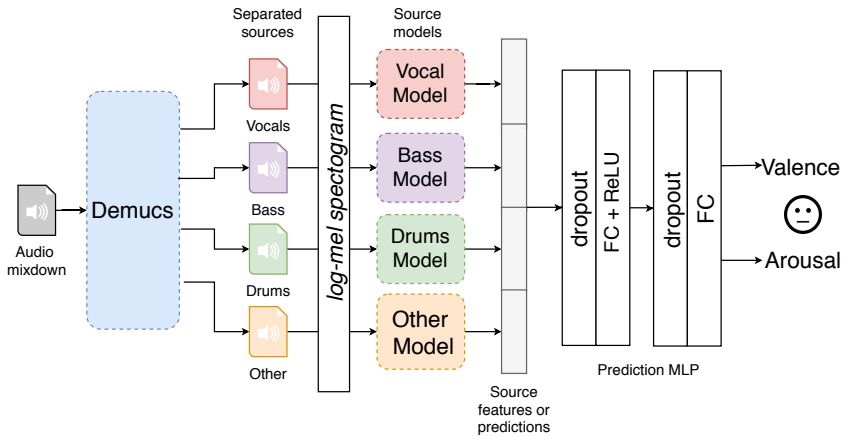


Figure 8.6: An overall architecture illustrating the proposed model, EmoMucs.

Music source separation module

In the final step of music production, the tracks corresponding to each individual instrument[1] are mixed together in a single audio file known as mix-down. Music source separation (MSS) aims at reconstructing the individual sources from a mix-down. A reference categorisation of these sources is the *SiSec Mus* evaluation campaign (Stöter, Liutkus, & Ito, 2018), which is based on the following classes: (i) *vocals*, (ii) *drums*, (iii) *bass* and (iv) *other*. Given a mix-down, the goal of a MSS model is to generate a waveform for each of the four original sources.

Most of the approaches for MSS operate on the spectrograms generated by the short-time Fourier transform (STFT). They are trained to produce a mask on the magnitude spectrums for each frame and source (Défossez, Usunier, Bottou, & Bach, 2019). The output audio is then obtained through an inverse STFT on the masked spectrograms, reusing the input mixture phase. However, a technical limitation of these approaches is the information loss resulting from the mix of sources, which cannot be easily recovered through masking.

For this reason we use *Demucs* (Défossez et al., 2019), a recent deep learning model for MSS directly operating on the raw input waveform. Instead of learning to generate a mask for each source, Demucs is inspired by models for music generation in the waveform domain. More precisely, it implements a U-net architecture with a convolutional encoder-decoder, and a bidirectional LSTM between them to increase the number of channels exponentially with depth – an approach originally proposed by Dauphin, Fan, Auli, and Grangier (2017).

Given an audio track, our system starts by feeding it to Demucs. This results into four different source tracks – one for each *SiSec Mus* class. To ensure comparability of our architecture with the baseline methods for MER, we compute the log-mel spectogram of each source track.

[1]The terms *voice*, *instrument* and *source* are used interchangeably.

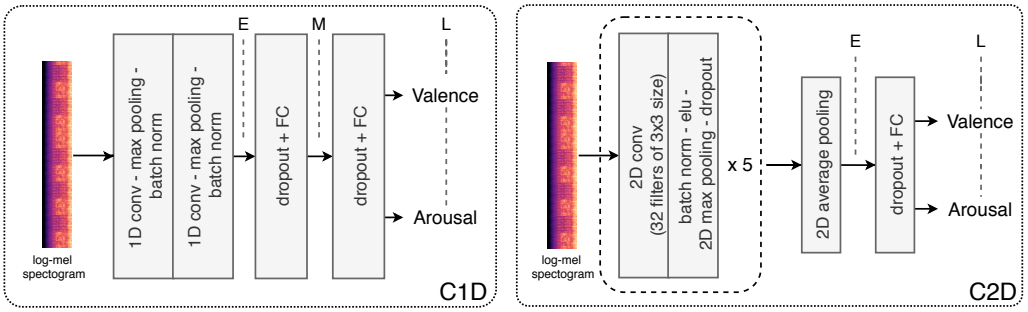


Figure 8.7: The baseline models in our system: a 1-dim (C1D) and a 2-dim convolutional neural network (C2D), implemented according to (Delbouys et al., 2018) and (Choi et al., 2017) respectively. In EmoMucs, these are used as the building blocks for the source models.

Source models and fusion strategies

As illustrated in Figure 8.6, the log-mel spectogram of each source track is then passed to the specific model associated to that source (e.g. the vocal's spectogram is fed to the vocal model). By disentanglement, each sub-model processes a single voice independently, and learns the corresponding source-specific musical features for emotion recognition. This approach thus provides a high degree of flexibility, as it makes it possible to design the architecture of each sub-model specifically for the corresponding source. Nonetheless, to ensure a fair comparison of our architecture with the current methods for MER, we simply use one of our baselines as the architecture for all source models.

The baseline models are based on two common deep learning architectures for MER, illustrated in Figure 8.7. The first is a one-dimensional convolutional neural network (C1D) that was proposed in (Delbouys et al., 2018) as an audio model for multimodal MER. The architecture consists of two one-dimensional convolutional layers followed by max-pooling and batch normalisation. Its resulting feature maps are then passed to two fully-connected layers with dropout masks to improve generalisation. The second baseline comprises a VGG-style network, demonstrated to be effective in several MIR tasks (Choi et al., 2017). This musically-engineered model, C2D, consists of 5 two-dimensional convolutional blocks, each separated by max pooling and dropout layers. The two-dimensional pooling operators progressively decrease the size of each feature map, while keeping the same number of kernels (32) after each block. After the convolutional blocks, two-dimensional average pooling is applied to ensure that the resulting feature map is of size 32×1. Following dropout, a single fully-connected layer (with no activation function) is then employed to predict arousal and valence.

The architecture of our source models can be either C1D or C2D, resulting in two different implementations of EmoMucs – *EmoMucs-C1D* and *EmoMucs-C2D*. To yield a final prediction of valence (V) and arousal (A), the features from the source models are concatenated and passed

to two fully-connected layers with dropout. Assuming C1D is chosen as the architecture for our source models, there are three possible ways to access the features of a source model. This gives three fusion strategies: *early* (E), *mid* (M) and *late* (L), depicted in Figure 8.7. The first strategy considers the features obtained after the convolutional layers; *mid* fusion concatenates the features learned after the first fully-connected layers; and the *late* strategy considers the output of each source model, which correspond to the VA predictions. From the definition of C2D, the *mid*-level fusion strategy is not possible with this architecture.

Our training approach

Considering the different role of each source model, there are three main strategies for training EmoMucs: *full*, *freeze* and *fine-tune*. The first approach (*full*) consists in training the whole network from scratch and propagating the gradient back to the source models from the last fully-connected layer of EmoMucs. In contrast, the last two strategies are based on pre-training each source model separately as a first step. The full network is then trained until the concatenation level, for the *freeze* mode, or until the first convolutional layer of each source model, for the *fine-tune* mode. This last choice can be considered as a sort of fine-tuning strategy and should be implemented with small learning rates to avoid the source models to catastrophically forget what they have already learned independently.

8.2.2 Experimental evaluation

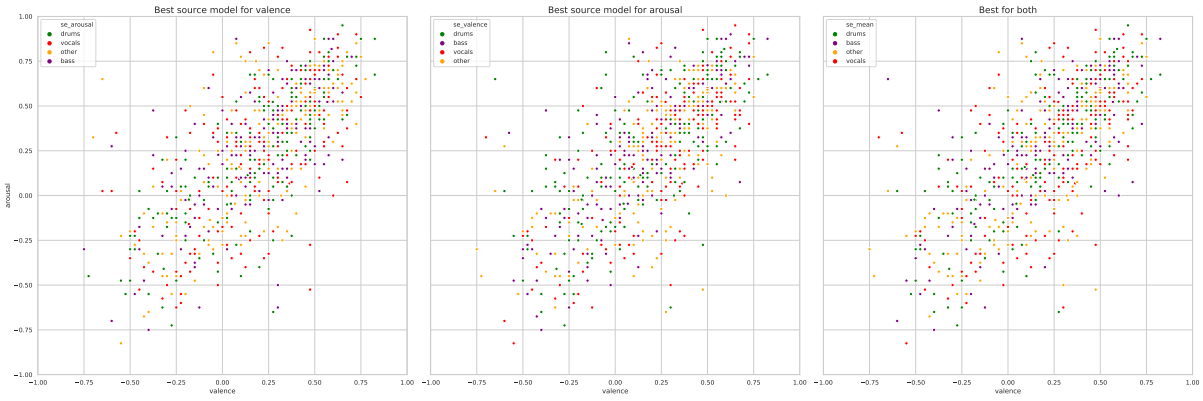


Figure 8.8: Target valence-arousal annotations of PMEmo in the $[-1, 1]$ interval, using colours to distinguish the C2D-based source model that better predicted each single annotation w.r.t. valence and arousal RMSE, together with their mean.

Our method is validated using the baseline models C1D and C2D trained on the mix-downs as reference models. These are denoted as C1D-M and C2D-M respectively. The performance

of the baselines is then compared with each source model trained independently. In particular, we compare CXD-M with CXD-$\{\text{V} \mid \text{B} \mid \text{D} \mid \text{O}\}$, where V, B, D, O denote the *vocals*, *bass*, *drum*, and *other* sources respectively, and $X \in \{1,2\}$. This allows to verify how informative each source model is, and whether one of them outperforms the correspondent mix-down baseline.

Secondly, we experiment with the different fusion and training strategies of EmoMucs using all the source models. Similarly to the previous case, the performances of EmoMucs with either C1D or C2D architectures for its source models (denoted as EmoMucs-C1D and EmoMucs-C2D) are compared to C1D-M and C2D-M. We evaluate the accuracy of the valence-arousal predictions using standard regression metrics: the root-mean-squared error (RMSE) and the coefficient of determination (the R^2 score), as also done in the previous section.

Dataset

As argued in (Saari et al., 2010; Song & Simon, 2015), current methods for MER can perform well for genres such as classical music and film soundtracks, but their performances are still poor for popular music. For this reason, we focus on pop music and use the *popular music with emotional annotations* (PMEmo) dataset K. Zhang et al. (2018) for our experiments. The collection contains valence-arousal induced emotions for 794 pop songs, annotated by 457 subjects, and also provides audio clips of the chorus sections (in MP3 format) together with pre-computed audio features at the song level.

For our experiments, we consider the static valence-arousal annotations. As EmoMucs needs raw-audio data to feed Demucs and generate the separated sources from a given mix-down, we use 20-second randomly selected clips from each chorus. For 59 tracks with duration shorter than 20 seconds, zero padding is applied at the end of the clip to ensure fixed-size input. On average, the padding operation is used to compensate for 4.35 seconds. The arousal and valence annotations are scaled to the $[-1,1]$ interval for improving the stability of the model. To conclude, we do not apply any data augmentation, as such strategies should be handled with care considering that they can potentially affect the emotional impact of music on listeners.

Implementation details

We use Librosa 0.7.2 (McFee, Lostanlen, McVicar, et al., 2020) for computing the log-mel spectograms from the tracks, with a fast Fourier transform (FFT) window size of 512, 256 samples between successive frames and 96 Mel bands. Our models are implemented in PyTorch (Paszke et al., 2019), and the source code to replicate these experiments is available at `github.com/jonnybluesman/emomucs`.

			RMSE		R2	
Baseline	# params	Input	V	A	V	A
C1D	86354	**M**	**.2600**	**.2444**	**.3489**	**.5573**
		V	.3048	.3214	.1131	.2426
		B	.2890	.3311	.2029	.1963
		D	.2710	.2961	.3000	.3572
		O	.2723	.2936	.2925	.3679
C2D	37698	**M**	**.2466**	**.2285**	**.4143**	**.6100**
		V	.2701	.2750	.3039	.4455
		B	.2762	.2924	.2718	.3732
		D	.2587	.2855	.3613	.4024
		O	.2633	.2748	.3381	.4462

Table 8.2: Evaluation of the baseline models trained on the mix-down (C1D-M, C2D-M) together with the corresponding source models. V, B, D, O denote *vocals*, *bass*, *drums* and *other* and they refer to the source models. Bold text highlights the best results for each baseline model. For both cases, the mix-down model achieves the best results.

		Early				Mid				Late			
		RMSE		R2		RMSE		R2		RMSE		R2	
Model	Training	V	A	V	A	V	A	V	A	V	A	V	A
EmoMucs-C1D	freeze	.2536	.2580	.3803	.5064	**.2428**	**.2435**	**.4332**	**.5615**	.2453	.2475	.4208	.5470
	finetune	.2562	.2624	.3655	.4878	.2516	.2492	.3875	.5395	na			
	full	.2536	.2628	.3787	.4850	.2625	.2651	.3371	.4794				
EmoMucs-C2D	freeze	.2373	**.2307**	.4584	**.6046**	na				**.2320**	**.2322**	**.4814**	**.6004**
	finetune	.2444	.2442	.4256	.5560					na			
	full	.2541	.2543	.3793	.5212								

Table 8.3: Comparison of EmoMucs models with different fusion and training strategies.

Experimental results

The results of our experiments are reported in Tables 8.2 and 8.3. From Table 8.2, we notice that the C2D architecture is more accurate than C1D in all scenarios. In addition, the former baseline is also a more parsimonious model, due to the less number of parameters. All the baselines trained on the mix-down are considerably better than the source models considered independently. For both the architectures, we find that the drums model is the best between the source models at predicting valence, whereas the bass model is the worst for arousal. The performance disagreement for the source models using different architectures (e.g. C1D-V and C2D-V) suggests that the convolutional architecture plays a crucial role for the type of musical features that can be learned. In fact, the 1-dim/temporal CNN was not as effective as the 2-dim CNN with small kernel size (VGG-style).

As can be seen from Table 8.3, combining all source models has a crucial impact on the performance of the model. We conjecture that this is achieved by the architecture of EmoMucs,

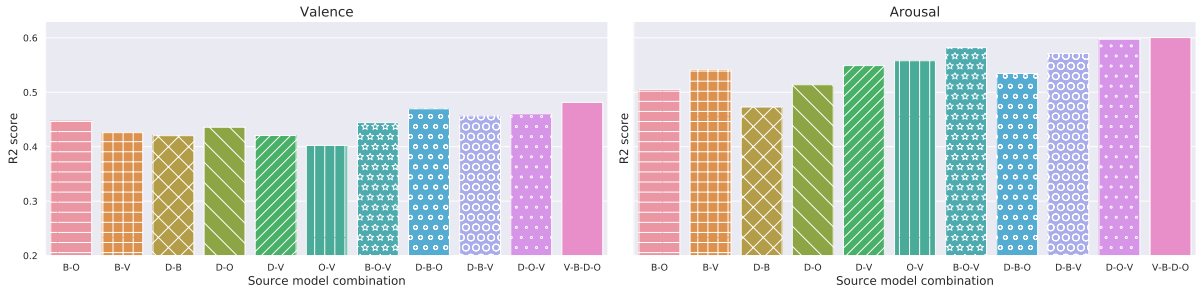


Figure 8.9: EmoMucs-C2D trained with different combinations of source models.

which makes it possible to account for all the possible inter-source relationships. In particular, EmoMucs-C1D with mid-level feature fusion and *freeze* mode training achieves a R^2 score of 0.4332 for valence, which is a considerable increase compared to 0.3489 for C1D. This is also reflected with a decrease of the RMSE for valence, accounting for 0.2428 instead of 0.26.

Considering that the valence-arousal annotations are scaled to the $[-1,1]$ interval, we divide these values by 2 for a more intuitive interpretation of the error. Hence, 0.2428 and 0.26 can be considered as errors of 12.14% and 13% in the annotation interval. Analogously, EmoMucs C2D with late fusion and *freeze* mode training achieves valence $R^2 = 0.4814$ and RMSE $=$ 0.2320 (11.6%), instead of $R^2 = 0.4143$ and RMSE $= 0.2466$ (12.33%) for the C2D baseline. On the other hand, the arousal predictions of EmoMucs are comparable to those of the baselines.

Interpretability

As the architecture of EmoMucs is based on a concatenation of features learned by each source model at a specific layer, it is possible to trace the contribution of each voice as well as those emerging from their interrelated connections. This form of interpretability is architecturally supported by our deep neural network, and it comes at no performance loss. This contrasts the work of Chowdhury et al. Chowdhury et al. (2019), who measured the "cost of explainibility" of their model by trading accuracy for interpretability.

A simple way to interpret EmoMucs is to isolate the performance of each model independently, as done in Table 8.2. It is also compelling to analyse the regression accuracy for each track in the dataset and visualise them together with the target annotations in the valence-arousal space. In Figure 8.8, this is done separately for valence and arousal by associating each target data point with a colour related to its best source model (the one with lowest valence and arousal RMSE for that target). If source models specialise in certain regions of the annotation space, e.g. drums and high arousal, we would expect them to form clusters in the annotation space. However, this hypothesis is rejected as Figure 8.8 does not suggest any clear specialisation of the source models in the annotation space. This supports our previous observation that each

track has intrinsic features related to its emotional impact. Two distinct tracks with very similar annotations can indeed have a considerably different emotional influence from their sources.

Figure 8.9 reports the performance (R^2 score for valence and arousal) of EmoMucs-C2D using late fusion and *freeze* training mode for different combinations of source models. The best performance is achieved when using all sources (V-B-D-O). The contribution of source models varies with their combination. When using two sources, the combination of the *bass* and the *other* models gives better performance in the valence space, but for the arousal space an improved result is achieved when combining *vocals* with *other*. When three sources are considered, the combination of *drums*, *bass* and *other* achieves the best R^2 for valence, whereas, for arousal, excluding *bass* gives comparable results to EmoMucs-C2D with all the sources.

8.3 Chapter summary

The task of computational music emotion recognition (MER) is particularly challenging due to several factors such as subjectivity within annotations, scarcity of labelled data for training supervised models, and inadequate data augmentation strategies. There is common belief that the current models perform well for classical music and film soundtracks, but their performances are still poor for popular music. To the best of our knowledge, improving the interpretability of MER models jeopardises their performance, thus introducing a "cost of explainability".

To address RQ 3, this last chapter investigated how structural properties of music can provide useful information for music emotion recognition, and more precisely, for the prediction of valence and arousal from audio tracks. This was done in two different ways using different structural properties of music: structures related to form (structures as nested musical segments), and instrumentation (structure emerging from the superimposition of sources).

First, the link between musical form and music emotions was explored, to see if the nested organisation of musical ideas in a performed composition has the potential to be relevant for predicting the emotional response of music listeners. For this purpose, a series of experiments evaluated the use of the structural complexity metrics introduced in Chapter 7 as audio features for music emotion recognition, with the main hypothesis being that these features can provide additive and independent information usable for this task. A feature selection experiment demonstrated the additive property, as a subset of structural features are retained for regression even when considered alongside state of the art audio features for MER. A randomisation experiment, instead, confirmed the independence of structural features, given that all models trained on the structural features for valence/arousal regression, always outperform their counterparts trained on a shuffled version of the same feature set.

Second, we focused on the link between instrumentation and music emotion recognition.

In particular, we introduced EmoMucs – a deep neural network that separates the audio signal into different sources (vocals, drums, bass and others), and uses different sub-models to process each source independently, with their features being aggregated at a later stage for regression. Through an extensive experimental evaluation including state of the art deep neural networks for MER on a pop music collection, EmoMucs was demonstrated to yield better recognition performance for valence, comparable performance for arousal, while opening the possibility to trace the contribution of sources for emotion recognition.

Part IV

Closing

Chapter 9

Conclusions

This thesis addressed the problem of automating the evaluation of music models, and the structural complexity of musical pieces. Through the analysis of a vast array of sequence models, this work first investigated how different modelling paradigms and dimensions can affect the performance of music models and the type of features they can learn. To measure the structural complexity of music, this thesis introduced a novel method for hierarchical music structure analysis and leveraged its output in such a way as to quantify structural properties by means of a list of metrics. Finally, the thesis also demonstrated that these structural metrics have the potential to provide complementary features for music emotion recognition (MER), where a link among structural complexity and music-induced emotions was conjectured in music psychology. Along the same line of research, another structural property of music that is not directly related to music form, was also leveraged for MER and found to improve the interpretability of models. In sum, these last contributions concluded the thesis while simultaneously opening a new research direction, where structure can guide, support, or improve the interpretability of models addressing other tasks in music information retrieval (MIR).

The experimental studies were presented in Chapter 5, Chapter 6, Chapter 7 and Chapter 8. In this chapter, Section 9.1 gives an overview of the milestones achieved in this thesis, with a summary of the contributions together with how the research questions were addressed. Section 9.2 motivates the design choices, together with highlighting their strengths and drawbacks, and Section 9.3 presents potential research directions that could expand and enhance the research work. This chapter closes with a few final remarks in Section 9.4.

9.1 Overview and contributions of thesis

Revisiting the contributions mentioned in Section 1.5, the research efforts will be explained in more technical detail now that the work has been fully presented. The primary goal of this thesis

is to advance research in music modelling and generation, focusing on the automatic evaluation of these methods with regards to their ability to learn structural properties, and generate music with a realistic level of structural complexity. The secondary goal of this work is to demonstrate that the development of new evaluation methods does not only benefit music modelling and generation, but has also the potential to support other tasks in music information retrieval. Following the presented contributions, the summary below outlines the effort collectively put to achieve the goals entailed by the research questions (c.f. Section 1.4).

Chapter 5 looked at RQ 1: "*How do different modelling paradigms and neural network architectures influence the performance of a music model learning structural properties from music?*" Specifically, the contributions were the following.

- To establish a standardised methodology for the systematic evaluation of music models, we introduced MOlliE, the first framework for music modelling providing a suite of utilities covering all the three main steps of the task: encoding of symbolic music, design of the architecture, and evaluation/analysis of the model. More precisely, the framework includes a collection of encoding functions to obtain vector representations from symbolic music, state-of-art neural network architectures, and analytical quantitative/qualitative methods allowing experimenters to design, evaluate, and compare their music models in a controlled environment. Instead of reinventing the wheel, novel architectures for music modelling can be easily integrated in the framework, benchmarked against other baselines, and made available to the research community.

- A novel architecture for music modelling – the long-short-term universal transformer (LSTUT), based on three different modelling paradigms: recurrence, induction, and self-attention. Music models based on the LSTUT achieved state of the art results on a large dataset, and managed to learn structural features directly in the attention heads. As argued in Section 2.1.2, these results demonstrate that the architecture has the potential to improve several tasks in music information retrieval, from automatic music transcription and style modelling to the generation of music with increased level of structural complexity.

- A large scale experiment leveraging the framework to evaluate the impact of the numerous modelling dimensions and design choices on the predictive capabilities of music models. In particular, this experiment allowed to analyse how different modelling paradigms, and neural network architectures influence the music modelling task in relation to the problem of learning structural properties, other than demonstrating the effectiveness of MOlliE.

Overall, the aforementioned contributions provided the computational infrastructure to run large scale music modelling experiments, focusing on the reusability and comparability of modelling approaches to evaluate new autoregressive architectures. The experiment carried out with

the framework, and the architectures found from the ablation studies, demonstrated that *recurrence*, *inductive bias*, and *attention* are all crucial modelling paradigms to learn structural features from symbolic music. This finding is mainly supported by the nature of the experimental setup (modelling long sequences with limited capacity) and the qualitative analysis of the LSTUT – the music model that achieved the best results in our experiments. In sum, this allowed to answer RQ 1. Nonetheless, further experiments are still needed to analyse the impact of different symbolic music encodings on the modelling task, which is left to future work.

Chapters 6 and 7 addressed RQ 2 – the main focus of this thesis: "*How can we formalise and measure the structural complexity of music in order to enable the automatic evaluation of AMC systems from a given set of generated compositions?*". Here, the two computational methods listed below were contributed.

- The dynamic music structure communities (DMSCOM) algorithm, a novel procedure for hierarchical music structure analysis (MSA) encoding an audio recording as a directed graph, and leveraging techniques for multi-resolution community detection to detect music structures at all possible levels. Our algorithm achieved state of the art hierarchical segmentation performance based on an in-depth analysis which also contributed a new method for evaluating MSA procedures, building on previous work on hierarchy expansion (McFee & Kinnaird, 2019). Compared to other algorithms in the literature, DMSCOM does not require particular hyper-parameter tuning, and its segmentation performance is less sensitive to the duration of tracks. The hierarchical segmentation it produces are also deeper, since it does not restrict or limit the size and type of segments to detect nor the topology of the hierarchies.

- By leveraging this last property of DMSCOM's structural segmentations, it was possible to introduce a set of metrics and a framework built on them to automatically measure the structural complexity of audio music. These metrics formally describe certain properties of these hierarchies and the decomposition of music structures therein estimated. Through an extensive experimental procedure including datasets of random, real and computer-generated music – groups which are expected to be associated with different degrees of structural complexity, it was found that not only do our measures permit to discriminate between them, but further non-trivial subdivisions can also be identified according to the structural properties of the compositions. Given the effectiveness of these metrics as structural descriptors of music, they were also demonstrated to be usable to evaluate automatic compositions systems from a repertoire of generations.

Both these contributions were crucial to initially align with the literature, and to ensure that

the implicit notion of structural complexity derived from the proposed metrics exclusively relates to music form, rather than encapsulating other musical properties that could have "leaked" from attempting to derive it from other sources. On one hand, the state of the art performance of DMSCOM on the hierarchical music structure analysis task, provides evidence that the estimated structural segmentations are reliable and consistent. On the other hand, the structural metrics extracted from the obtained segmentations were demonstrated to provide a continuum – from pseudo-random (little or no structure) to real music (structure at all levels), where the output of music generation systems can be effectively and efficiently evaluated for structural complexity. All of these results allowed to fully answer RQ 2.

Chapter 8 looked at the last research goal, which is formalised through RQ3: "*Could we improve the interpretability and the performance of music emotion recognition (a specific task in MIR) by leveraging the availability of music structural properties?*". To that aim, two methods focused on different structural properties – form and instrumentation, were contributed.

- An investigation on the use of the structural complexity metrics as audio features for music emotion recognition (MER). For this study, music recordings from a well-know MER dataset for valence/arousal regression were considered. From these tracks, state of the art audio features for MER were extracted, together with the structural complexity metrics. Based on a feature selection experiment, it was found that some of the latter are not discarded for regression – meaning that they can provide additive information in comparison to traditional audio features for MER. Another experiment looked at using the structural complexity metrics as the only input features for valence/arousal regression. From this, all models using this setup were found to still outperform baselines regressors, thereby providing further evidence to validate the previous finding.

- The Emotion Multiplexer (EmoMucs) – a novel architecture for MER based on the separation of musical sources to provide increased interpretability. This was achieved through the design of a deep neural network combining methods for music source separation with state of the art architectures for MER. The music signal first undergoes a source separation step, where different spectograms corresponding to *voice*, *bass*, *guitar* and *drums* are extracted, then fed to separate modules to learn source-specific features. The output of these modules is then aggregated by concatenation and finally passed to a multi-layer perceptron for valence/arousal prediction. Through a series of experiments on a MER dataset of pop music, results demonstrated that EmoMucs outperformed state of the art MER models for arousal, gave comparable results for valence, while allowing the inspection of the model to trace the contribution of sources.

Although these last contributions are based on preliminary studies in music emotion recognition, the obtained results still validated the potential of leveraging structural properties of music to improve computational methods for music analysis – thereby opening a new research direction where structurally-informative descriptors and features can guide and support computational models addressing MIR tasks, other than increasing their interpretability.

9.2 Critical analysis

Although the contributions of this thesis represent novel discoveries in their kind, the methods underneath have some technical limitations which may limit their current applicability to certain use cases. Being aware of these weaknesses is key to control their use depending on the target application domains, and also provides valid material and motivations for further research.

The first critical observation does not pertain to any technical aspect of this work, but is more profound due to the assumptions on the type of music this thesis focuses on. More precisely, our definition of music structure, and in particular of music form, can be appropriate when considering Western tonal music, but may be incorrect or inconsistent for other music, such as traditional Indian music, Japanese folk, or art music of the Near- and Middle- East. For instance, non-Western music tend to be more freely-structured and heavily reliant on open-ended improvisation. In this regard, the choice to focus on Western tonal music is motivated by the increased familiarity of the authors, and also considering the availability of datasets in music information retrieval research covering this type of music.

Concerning the more technical limitations, it is now possible to motivate the critical design choices of the proposed methods and highlight the main drawbacks that were found.

The evaluation toolbox provided in MOlliE – the new music modelling framework, currently permits to analyse statistical properties of the test loss, and perform qualitative analysis of the attention heads in search for structural patterns. By definition, this last kind of analysis can only be carried out on those music models whose architecture include an attention mechanism. This means that it is currently not possible to explore the feature space learned by non-attention layers. In fact, whereas inspecting the way attention distributes over sequence elements can unveil structural properties of a signal, the same level of interpretability cannot be so easily obtained when inspecting a fully-connected layer. This is definitely a strength of attention mechanisms, which are also recognised as a tool for explainability.

In addition, when looking at the quantitative side of the evaluation, comparing the cross entropy loss to explain the structural capabilities of a music model may sound like an over-simplifying methodology. Nevertheless, the experiments tested the predictive capabilities of music models by stressing them on considerably long sequences while keeping their memory

capacity relatively low. Under these settings, with musical sequences containing more than 5K tokens, a sequence model using a limited number of parameters would potentially learn to relate musical patterns at different time scales, rather than attempting to memorise all sequence elements – something that would be beyond its memory capacity. For instance, a possible way to do this is by detecting *repetitions* and *variations*, both of which are key elements for identifying the temporal boundaries and the structural groups of music segments (the staple tasks in music structure analysis). Therefore, using long sequences and small models should provide a reasonable experimental setup to preliminary test the structural capabilities of music models.

Following the order of contributions, another critical point concerns the output produced by DMSCOM – the proposed state of the art algorithm for hierarchical music structure analysis. Although DMSCOM produces rich structural segmentations of music, containing several layers that progressively refine structures identified at coarser levels, the considerable depth of these hierarchies can be seen as a double-edged sword. On one hand, they are particularly informative to be processed and analysed computationally, as indeed demonstrated with the structural complexity metrics. On the other hand, the large number of levels these hierarchies may contain can also complicate their manual inspection. This is merely due to the specific design choice of avoiding any potential bias in the segmentation process. In fact, it was desirable not to restrict or limit the size and type of structures to detect nor the topology of the hierarchies. Nevertheless, to make the output of DMSCOM useful to musicologists, the estimated hierarchies should undergo a post-processing step based on genre/style-dependent heuristics that can reduce the number of undesired layers. In this way, only the relevant segmentation layers would be retained and the resulting hierarchies would be more compact to be manually inspected.

Concerning the structural complexity metrics derived from hierarchical segmentations of music, one limitation of this method is the lack of a musically plausible formulation of these metrics. Indeed, the structural complexity metrics were designed to formally describe both static and dynamic properties of these hierarchical segmentations, which are thus considered as mathematical objects. Therefore, it is not directly possible to relate the output of these measures with certain musical properties that could explain the level and type of structural complexity of a musical piece. The reason behind this matter is again due to a design choice. Instead of attempting a subjective and musically plausible definition of structural complexity – something that would be ambiguous and could potentially raise several concerns in music research, the proposed approach is based on the hypothesis that the former is a latent property of music structure segmentations. Describing specific attributes of these hierarchical segmentations thus provides a collection of compact structural descriptors that can summarise these properties.

The last critical remarks concern the use of the structural complexity metrics to support research in music information retrieval. As demonstrated in Section 8.1, the structural complexity

metrics can provide additive information for MER when used in conjunction with state of the art audio features. However, from the experiments it was also found that this setup still does not significantly improve the performance of vanilla MER models trained on the audio features only. This means that the use of the structural complexity metrics as additional input features has potential use for MER, although no regressor was yet capable to exploit them to make more accurate predictions for valence and arousal. One possible explanation for this is that the new feature space introduces more complex decision boundaries that cannot be easily learned by traditional regression models. In this case, multi-layer perceptrons or more complex neural network architectures (such as those presented in Section 8.2) could potentially learn such highly non-linear transformations and yield better results. Nevertheless, traditional machine learning models for valence/arousal regression were preferred for this preliminary study, as summative audio features for static MER are typically used with these models.

9.2.1 Overview of music representations in this thesis

When looking at the models, algorithms, and tools developed in this research project, only MOlliE (c.f. Chapter 5) expects symbolic music as input, whereas all the remaining contributions are defined on audio music. This is due to the abundance of music models in the symbolic domain, and to the fact that the problem of learning structural properties from music – at all possible levels, and consequently, the challenge of generating music endowed with a realistic level of structural complexity, can already be traced from the symbolic domain. Indeed, as motivated in Section 1.2 and extensively argued in the literature review (Chapter 4), symbolic music models notoriously struggle to achieve this goal, and the problem is further exacerbated in the audio domain, where sequences are considerably longer to process. Therefore, focusing on modelling (predicting) symbolic music should be viewed as a first step towards the generation of realistic scores from the resulting models. Lessons learned from such research efforts could then potentially be brought to and adapted to audio music modelling – a more difficult challenge.

On one hand, MOlliE addresses the music modelling task – where the problem of learning structural dependencies stems from, focusing on symbolic music as per the aforementioned reasons. On the other hand, DMSCOM (c.f. Chapter 6) and the structural complexity metrics (c.f. Chapter 7) are audio-based methods for two main reasons: (i) the literature in music structure analysis is more well-studied in the audio domain, where structurally-annotated datasets and state of the art segmentation methods are available for use and comparison; (ii) the scope of RQ 2 is more general, as we seek to address the evaluation of structural complexity from arbitrary music recordings – regardless of the underlying generation process (e.g. autoregressive music models, rule-based systems, semi-automatic methods, template-filling algorithms) and format. In fact, given the diversified nature of the experimental setup (c.f. Section 7.2),

including synthesised MIDI files and live music recordings, the evaluation framework based on the structural complexity metrics was demonstrated to be effective for both audio music and symbolic music (after sonification of MIDI files).

Analogously, both contributions in Chapter 8 are audio-based due to the availability of audio datasets in MER, and the methodological reason that music-induced emotions are collected/annotated and detected from music recordings. In contrast, symbolic-based MER models are generally trained to detect perceived emotions from music scores (c.f. Section 2.3.1).

9.3 Future directions

"*In completing one discovery we never fail to get an imperfect knowledge of others.*"
(Joseph Priestley)

Overall, the central goal of this work is to introduce the first computational methods that can be used to systematically evaluate the structural complexity of music. Although structural complexity can potentially support different application domains (c.f. Section 8.1), the contributed methods were originally conceived for automatically evaluating the output of music generation systems. Nonetheless, in line with many academic endeavours, this research work also opens up several windows for further questions, leading to the proposition of various subsequent research avenues. Among these, we plan to pursue the following research directions.

9.3.1 Structure-aware music modelling and generation

As this work demonstrated the effectiveness of the structural complexity metrics for evaluating music generation systems, we plan to investigate their use to also improve the output of such algorithmic procedures. More precisely, these metrics can be used in conjunction with a validation strategy to promote the structural complexity of the resulting generations, or considered aposteriori to fine-tune a music model after training. Both these approaches share the same intent – improving the structural capabilities of a music generation system by leveraging our metrics to systematically evaluate the structural complexity of a repertoire of generated compositions. Either way, the metrics would thus be used as a form of *structural supervision*.

Considering that the computation of these metrics is not differentiable per se, and requires sampling sequences from a music model (another non-differentiable operation), the structural supervision cannot be directly implemented in the training loss. As mentioned before, a first option is to redefine the validation loss to include a structural regularisation term – which will tend to zero when the structural complexity of the epoch's generations falls within the area of human-composed music. This is already encoded by the Kolmogorov-Smirnov score between

the generations' distribution and that of the human reference group in the structural summaries space (c.f. Section 7.3). In doing so, the training process can be steered in such a way as to encourage exploration (the stochastic components of the model, e.g. dropout rates) whenever the structural loss shows an increasing behaviour or a stagnation trend.

Alternatively, structural supervision can also be provided by fine-tuning a pre-trained music model. Instead of intervening on the training strategy, the parameters of a music model can be progressively adjusted by means of a reinforcement learning (RL) procedure. In this settings, the trained music model would play the role of the agent, and the learning objective is to find an optimal configuration of the network maximising the structure-guided return – reflecting both the structural metrics and information learned from data. This is now possible as RL can allow a network to learn a non-differentiable reward function. During each episode, a number of compositions would be sampled from the model, and their structural summaries computed as detailed in Section 7.3. Analogously to the previous approach, the negation of the Kolmogorov-Smirnov score between the generations' structural summaries and those of the human reference group can then be used as a *structural reward*. After each episode, the model should thus be able to generate music with a more realistic level of structural complexity. A similar RL-based technique was already investigated by Jaques et al. (2017), although their method focuses on specific music theory rewards to avoid excessive repetition in the generations, while encouraging tonal and harmonic consistency.

9.3.2 Automating the evaluation of music generation models

With MOlliE and the structural complexity metrics, this thesis contributed to the automatic evaluation of computational methods for music modelling and generation, focusing on music structure. Nonetheless, the evaluation of these methods can only be exhaustive if it considers more musical dimensions – including structural complexity. For instance, it is common to seek clarification on whether generated music is theoretically plausible in regard to harmony and melody, but also to quantify other properties such as tonal and rhythmic complexity, and the memorability of a piece. As reiterated in the previous sections, such a detailed evaluation is still done manually by music experts and is commonly adopted for music generation challenges.

To improve accessibility and reproducibility, the next challenge is to fully automate this complex and multi-faceted evaluation process. More precisely, this future research direction will lead to a comprehensive suite of computational tools providing a fast, controlled, and reproducible methodology for the systematic evaluation of music generation systems. Regardless of their technical implementation, a repertoire of compositions will be generated from these systems and fed to our evaluation utilities – each describing and/or quantifying a specific musical

property of the given musical material. In this way, users will be able to automatically analyse the output of their music generation systems, and visualise where each musical dimension stands in comparison to real/human compositions (of different styles and genres).

From a global perspective, this will allow to align the work on structural complexity with the recent endeavours in automatic music evaluation (L.-C. Yang & Lerch, 2020), while complementing it with the current *music metrics* in music psychology and information retrieval research. These include, but are not limited to, measures of tonal (Di Giorgi, Dixon, Zanoni, & Sarti, 2017; Weiss & Müller, 2015), harmonic (Mihelač & Povh, 2020) and rhythmic (Toussaint & Trochidis, 2018) complexity of music, as well as properties related to musicality (Pease, Mahmoodi, & West, 2018) and individuality (Wöllner, 2013) of performances. As these methods cannot be taken as they are out-of-the-box, we will first investigate their suitability in the context of music evaluation, then integrate them within our computational suite.

9.4 Closing remarks

The design of novel methods for computational music analysis, enabling the automatic extraction of numerical descriptors of specific musical properties – from harmonic, rhythmic to structural complexity, has a broad array of potential applications. For instance, these measures can support several tasks in music information retrieval, including search and exploration of large music collections and repertoires, music recommendation, automatic music tagging, and also simplify the manual workflow of musicological studies. Another potential application is for evaluating music generated by automatic composition systems. This last direction was central to this thesis, as it addresses a specific gap in the literature and encourages future endeavours.

The automatic generation of music that cannot be distinguished from human compositions still remains an open challenge for machine learning and machine creativity research. Nonetheless, I believe that improving the evaluation arsenal for music generation systems, while jointly devising better methods for sequence modelling, can bring us closer to this goal. The former will allow a thorough analysis of these systems by providing a set of computational tools to: (i) inspect the feature space learned by these models in search for musically plausible explanations; and (ii) review the machine-generated repertoires to measure "how good/realistic" their compositions are in relation to specific musical properties. Not only do these evaluation methods enable the systematic comparison of music generation systems, but the insights of such analyses can also inform and drive the design of new sequence models for music – the common backbone of these systems. For this last goal, research in machine learning and natural language processing can be leveraged and harmonised in the music domain, as the properties of the music signal per se pose a number of non-trivial challenges that deserve individual treatment.

In the meantime, it is also worth opening a new debate on the eventual implications of these methods, as having a system that can fully generate realistic music inevitably raises several concerns from an ethical perspective. In this regard, my research distances from all those methods and services that intend to fully automate the creative process of music composition – a form of human artistic expression. Instead of replacing artists and composers, research should focus on leveraging the generative capabilities of these systems to design new interfaces that can support, enhance and augment the creative potential of composers. To name but a few, this includes the generation of original musical ideas that can help overcome "composer's block", the use of style transfer methods to adapt a composition to a different genre-style, the benefit of music inpainting to complete the "missing parts" of a composition, and the automatic generation of arrangements. The resulting human-machine collaboration is an example of Artificial Intelligence Augmentation (AIA), an emerging field aimed at using Artificial Intelligence (AI) systems to develop new methods for Intelligence Augmentation (IA), as originally formulated by Carter and Nielsen (2017). While augmenting the creative capabilities of music professionals, the generative capabilities of these systems can also be useful to unskilled practitioners. In the near future, even an amateur musician with no fundamental notions of music theory will be able to compose new musical pieces, a direction that opens the door to intelligent computer-assisted tools for music education.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from `https://www.tensorflow.org/` (Software available from tensorflow.org)

Aljanaki, A., & Soleymani, M. (2018). A data-driven approach to mid-level perceptual musical feature modeling. *arXiv preprint arXiv:1806.04903*.

Aljanaki, A., Yang, Y.-H., & Soleymani, M. (2017a). Developing a benchmark for emotional analysis of music. *PloS one*, *12*(3), e0173392.

Aljanaki, A., Yang, Y.-H., & Soleymani, M. (2017b, 03). Developing a benchmark for emotional analysis of music. *PLOS ONE*, *12*, e0173392. doi: 10.1371/journal.pone.0173392

Allan, M., & Williams, C. (2005). Harmonising chorales by probabilistic inference. In *Advances in neural information processing systems* (pp. 25–32).

Amso, D., & Scerif, G. (2015). The attentive brain: insights from developmental cognitive neuroscience. *Nature Reviews Neuroscience*, *16*(10), 606–619.

Angeline, P. J., Saunders, G. M., & Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, *5*(1), 54–65.

Arenas, A., Fernandez, A., & Gomez, S. (2008). Analysis of the structure of complex networks at different resolution levels. *New Journal of Physics*, *10*(5), 053039.

Ariza, C. (2009). The interrogator as critic: The turing test and the evaluation of generative music systems. *Computer Music Journal*, *33*(2), 48–70.

Arjovsky, M., Shah, A., & Bengio, Y. (2016). Unitary evolution recurrent neural networks. In *International conference on machine learning* (pp. 1120–1128).

Association, M. M., et al. (2005). Midi specifications. *URL: http://www. midi. org/about-midi/specshome. shtml*.

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Baddeley, A. (1992). Working memory. *Science*, *255*(5044), 556–559.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to

align and translate. *arXiv preprint arXiv:1409.0473*.

Bai, S., Kolter, J. Z., & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

Barthet, M., Fazekas, G., & Sandler, M. (2012). Music emotion recognition: From content- to context-based models. In *Cmmr*.

Bartsch, M. A., & Wakefield, G. H. (2005). Audio thumbnailing of popular music using chroma-based representations. *IEEE Transactions on Multimedia*, *7*(1), 96–104.

Bay, M., Ehmann, A. F., & Downie, J. S. (2009). Evaluation of multiple-f0 estimation and tracking systems. In *Ismir* (pp. 315–320).

Bayer, J., Osendorfer, C., Korhammer, D., Chen, N., Urban, S., & van der Smagt, P. (2013). On fast dropout and its applicability to recurrent networks. *arXiv preprint arXiv:1311.0701*.

Bengio, Y., et al. (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, *2*(1), 1–127.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, *5*(2), 157–166.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*(Feb), 281–305.

Bernd, K. (1996). *Webpage of the pianomidi dataset.* `http://www.piano-midi.de/`. (Accessed: 2020-07-02)

Bertin-Mahieux, T., Ellis, D. P., Whitman, B., & Lamere, P. (2011). The million song dataset. In *Ismir* (Vol. 2, p. 10).

Bloice, M. D., Stocker, C., & Holzinger, A. (2017). Augmentor: An image augmentation library for machine learning. *arXiv preprint arXiv:1708.04680*.

Bodarwé, K.-A., Noack, J., & Jean-Jacques, P. (2015). Emotion-based music recommendation using supervised learning. In (p. 341–344). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/2836041.2841204` doi: 10.1145/2836041.2841204

Boulanger-Lewandowski, N., Bengio, Y., & Vincent, P. (2012). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*.

Briot, J.-P., Hadjeres, G., & Pachet, F. (2017). Deep learning techniques for music generation - a survey.

Briot, J.-P., Hadjeres, G., & Pachet, F.-D. (2020). *Deep learning techniques for music generation*. Springer.

Bruderer, M. J., McKinney, M. F., & Kohlrausch, A. (2006). Structural boundary perception in popular music. In *Proceedings of the 7th international conference on music information*

retrieval (pp. 198–201).

Brunner, G., Wang, Y., Wattenhofer, R., & Wiesendanger, J. (2017). Jambot: Music theory aware chord based generation of polyphonic music with lstms. *arXiv preprint arXiv:1711.07682*.

Camilus, K. S., & Govindan, V. (2012). A review on graph based segmentation. *International Journal of Image, Graphics & Signal Processing*, *4*(5).

Carnovalini, F., & Rodà, A. (2020). Computational creativity and music generation systems: An introduction to the state of the art. *Frontiers in Artificial Intelligence*, *3*, 14.

Carter, S., & Nielsen, M. (2017). Using artificial intelligence to augment human intelligence. *Distill*. (https://distill.pub/2017/aia) doi: 10.23915/distill.00009

Cho, K., Courville, A., & Bengio, Y. (2015). Describing multimedia content using attention-based encoder-decoder networks. *IEEE Transactions on Multimedia*, *17*(11), 1875–1886.

Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Choi, K., Fazekas, G., Sandler, M. B., & Cho, K. (2017). Transfer learning for music classification and regression tasks. *ArXiv*, *abs/1703.09179*.

Chowdhury, S., Vall, A., Haunschmid, V., & Widmer, G. (2019). Towards explainable music emotion recognition: The route via mid-level features. In *Ismir*.

Chu, H., Urtasun, R., & Fidler, S. (2016). Song from pi: A musically plausible network for pop music generation. *arXiv preprint arXiv:1611.03477*.

Chuan, C.-H., & Herremans, D. (2018). Modeling temporal tonal relations in polyphonic music through deep networks with a novel image-based representation.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Clauset, A., Newman, M. E., & Moore, C. (2004). Finding community structure in very large networks. *Physical Review E*, *70*(6), 066111.

Collins, N. (2009). Musical form and algorithmic composition. *Contemporary Music Review*, *28*(1), 103–114.

Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç., & Courville, A. (2016). Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*.

Cooper, G., & Meyer, L. B. (1963). *The rhythmic structure of music*. University of Chicago Press.

Corradini, F., De Angelis, F., Polini, A., Castagnari, C., de Berardinis, J., & Forcina, G. (2020). Tangramob: an agent-based simulation framework for validating urban smart mobility solutions. *Journal of Intelligent Systems*, *29*(1), 1188–1201.

Coutinho, E., Alshukri, A., de Berardinis, J., & Dowrick, C. (2021). Polyhymnia mood – empowering people to cope with depression through music listening. In *Adjunct proceedings of the 2021 acm international joint conference on pervasive and ubiquitous computing and proceedings of the 2021 acm international symposium on wearable computers* (p. 188–193). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3460418.3479334 doi: 10.1145/3460418.3479334

Coutinho, E., & Schuller, B. (2021). The polyhymnia acoustic and musical feature set: a new mer specific feature set. *Manuscript in preparation*.

Coutinho, E., Trigeorgis, G., Zafeiriou, S., & Schuller, B. (2015, 01). Automatically estimating emotion in music with deep long-short term memory recurrent neural networks. In (p. 1-3).

Culurciello, E. (n.d.). *The fall of rnn and lstm.* https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0.

Dannenberg, R. B. (2005, September). Toward automated holistic beat tracking, music analysis and understanding. In *Proceedings of the 6th international conference on music information retrieval* (p. 366-373). Retrieved from https://doi.org/10.5281/zenodo.1415246 doi: 10.5281/zenodo.1415246

Dauphin, Y. N., Fan, A., Auli, M., & Grangier, D. (2017). Language modeling with gated convolutional networks. In *Proceedings of the 34th international conference on machine learning-volume 70* (pp. 933–941).

de Berardinis, J., Barrett, S., Cangelosi, A., & Coutinho, E. (2020). Modelling long-and short-term structure in symbolic music with attention and recurrence. In *Proceedings of the 2020 joint conference on ai music creativity* (pp. 1–11).

de Berardinis, J., Cangelosi, A., & Coutinho, E. (2020). The multiple voices of musical emotions: Source separation for improving music emotion recognition models and their interpretability. In *Proceedings of the 21st international society for music information retrieval conference* (pp. 310–317).

de Berardinis, J., Cangelosi, A., & Coutinho, E. (2022). Measuring the structural complexity of music: from structural segmentations to the automatic evaluation of models for music generation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing (manuscript accepted)*.

de Berardinis, J., Forcina, G., Jafari, A., & Sirjani, M. (2018). Actor-based macroscopic

modeling and simulation for smart urban planning. *Science of Computer Programming*, *168*, 142–164.

de Berardinis, J., Pizzuto, G., Lanza, F., Chella, A., Meira, J., & Cangelosi, A. (2020). At your service: Coffee beans recommendation from a robot assistant. In *Proceedings of the 8th international conference on human-agent interaction* (pp. 257–259).

de Berardinis, J., Vamvakaris, M., Cangelosi, A., & Coutinho, E. (2020). Unveiling the hierarchical structure of music by multi-resolution community detection. *Transactions of the International Society for Music Information Retrieval*, *3(1)*, 82–97.

Défossez, A., Usunier, N., Bottou, L., & Bach, F. (2019). Music source separation in the waveform domain. *arXiv preprint arXiv:1911.13254*.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., & Łukasz Kaiser. (2018). *Universal transformers.*

Delbouys, R., Hennequin, R., Piccoli, F., Royo-Letelier, J., & Moussallam, M. (2018). Music mood detection based on audio and lyrics with deep neural net. In *Ismir.*

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dhariwal, P., Jun, H., Payne, C., Kim, J. W., Radford, A., & Sutskever, I. (2020). Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*.

Dieleman, S. (2015). *Learning feature hierarchies for musical audio signals* (Unpublished doctoral dissertation). University of Ghent.

Dieleman, S., Oord, A. v. d., & Simonyan, K. (2018). The challenge of realistic music generation: modelling raw audio at scale. *arXiv preprint arXiv:1806.10474*.

Di Giorgi, B., Dixon, S., Zanoni, M., & Sarti, A. (2017). A data-driven model of tonal chord sequence complexity. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *25*(11), 2237–2250.

Dornbush, S., Fisher, K., McKay, K., Prikhodko, A., & Segall, Z. (2005). Xpod - a human activity and emotion aware mobile music player. In *2005 2nd asia pacific conference on mobile technology, applications and systems* (p. 1-6).

Dozat, T. (2016). Incorporating nesterov momentum into adam.

Dubnov, S., Assayag, G., Lartillot, O., & Bejerano, G. (2003). Using machine-learning methods for musical style modeling. *Computer*, *36*(10), 73–80.

Duch, J., & Arenas, A. (2005). Community detection in complex networks using extremal optimization. *Physical Review E*, *72*(2), 027104.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(Jul), 2121–2159.

Eck, D., & Lapalme, J. (2008). Learning musical structure directly from sequences of music.

University of Montreal, Department of Computer Science, CP, *6128*, 48.

Eck, D., & Schmidhuber, J. (2002a). Finding temporal structure in music: Blues improvisation with lstm recurrent networks. In *Proceedings of the 12th ieee workshop on neural networks for signal processing* (pp. 747–756).

Eck, D., & Schmidhuber, J. (2002b). A first look at music composition using lstm recurrent neural networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, *103*.

Eck, D., & Schmidhuber, J. (2002c). Learning the long-term structure of the blues. In *International conference on artificial neural networks* (pp. 284–289).

Eerola, T., Friberg, A., & Bresin, R. (2013). Emotional expression in music: contribution, linearity, and additivity of primary musical cues. *Frontiers in psychology*, *4*, 487.

Ekman, P. (1992). An argument for basic emotions..

Elliot, W. (2016). *Generating Long-Term Structure in Songs and Stories.* `https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn/`. (Accessed: 2020-07-02)

Fedorenko, E., McDermott, J. H., Norman-Haignere, S., & Kanwisher, N. (2012). Sensitivity to musical structure in the human brain. *Journal of Neurophysiology*, *108*(12), 3289–3300.

Fiebrink, R., Caramiaux, B., Dean, R., & McLean, A. (2016). *The machine learning algorithm as creative musical tool*. Oxford University Press.

Foote, J. (1999). Visualizing music and audio using self-similarity. In *Proceedings of the seventh acm international conference on multimedia (part 1)* (pp. 77–80).

Foote, J. (2000). Automatic audio segmentation using a measure of audio novelty. In *Ieee international conference on multimedia and expo* (Vol. 1, pp. 452–455).

Fortunato, S., & Barthelemy, M. (2007). Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, *104*(1), 36–41.

Franklin, J. A. (2006). Recurrent neural networks for music computation. *INFORMS Journal on Computing*, *18*(3), 321–338.

Gabrielsson, A. (2001). Emotion perceived and emotion felt: Same or different? *Musicae scientiae*, *5*(1_suppl), 123–147.

Gaver, W. W., & Mandler, G. (1987). Play it again, sam: On liking music. *Cognition and Emotion*, *1*(3), 259-282. doi: 10.1080/02699938708408051

Gers, F. A., & Schmidhuber, J. (2000). Recurrent nets that time and count. In *ijcnn* (p. 3189).

Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with lstm.

Goel, K., Vohra, R., & Sahoo, J. (2014). Polyphonic music generation by modeling temporal dependencies using a rnn-dbn. In *International conference on artificial neural networks* (pp. 217–224).

Goetschius, P. (1904). *Lessons in music form: A manual of analysis of all the structural factors and designs employed in musical composition*. Oliver Ditson Company.

Goldman-Rakic, P. S. (1995). Cellular basis of working memory. *Neuron*, *14*(3), 477–485.

Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., & Sculley, D. (2017). Google vizier: A service for black-box optimization. In *Proceedings of the 23rd acm sigkdd international conference on knowledge discovery and data mining* (pp. 1487–1495).

Gómez, E. (2006). *Tonal description of music audio signals* (Doctoral dissertation, Universitat Pompeu Fabra). doi: http://hdl.handle.net/10803/7537

Gomez, P., & Danuser, B. (2007). Relationships between musical structure and psychophysiological measures of emotion. *Emotion*.

Good, M. (2001). Musicxml for notation and analysis. *The virtual score: representation, retrieval, restoration*, *12*(113-124), 160.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. (`http://www.deeplearningbook.org`)

Goto, M. (2006). A chorus section detection method for musical audio signals and its application to a music listening station. *IEEE Transactions on Audio, Speech, and Language Processing*, *14*(5), 1783–1794.

Goto, M., & Dannenberg, R. B. (2018). Music interfaces based on automatic music signal analysis: new ways to create and listen to music. *IEEE Signal Processing Magazine*, *36*(1), 74–81.

Goto, M., Yoshii, K., Fujihara, H., Mauch, M., & Nakano, T. (2011). Songle: A web service for active music listening improved by user contributions. In *Proceedings of the 12th international society for music information retrieval conference* (pp. 311–316).

Granell, C., Gomez, S., & Arenas, A. (2012). Hierarchical multiresolution method to overcome the resolution limit in complex networks. *International Journal of Bifurcation and Chaos*, *22*(07), 1250171.

Graves, A. (2011). Practical variational inference for neural networks. In *Advances in neural information processing systems* (pp. 2348–2356).

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Graves, A. (2016). Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

Graves, A., Mohamed, A.-r., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on* (pp. 6645–6649).

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional

lstm and other neural network architectures. *Neural Networks*, *18*(5-6), 602–610.

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2017). Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, *28*(10), 2222–2232.

Grill, T., & Schlüter, J. (2015). Music boundary detection using neural networks on combined features and two-level annotations. In *Proceedings of the 16th international society for music information retrieval conference* (pp. 531–537).

Gulati, S., Serra, J., Ishwar, V., & Serra, X. (2016). Discovering rāga motifs by characterizing communities in networks of melodic patterns. In *Ieee international conference on acoustics, speech and signal processing* (pp. 286–290).

Hadjeres, G., Pachet, F., & Nielsen, F. (2016). Deepbach: a steerable model for bach chorales generation. *arXiv preprint arXiv:1612.01010*.

Hawthorne, C., Stasyuk, A., Roberts, A., Simon, I., Huang, C.-Z. A., Dieleman, S., ... Eck, D. (2019). Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International conference on learning representations.* Retrieved from https://openreview.net/forum?id=r1lYRjC9F7

Hermans, M., & Schrauwen, B. (2013). Training and analysing deep recurrent neural networks. In *Advances in neural information processing systems* (pp. 190–198).

Herremans, D., Chuan, C.-H., & Chew, E. (2017). A functional taxonomy of music generation systems. *ACM Computing Surveys (CSUR)*, *50*(5), 1–30.

Hiller, L. A., & Isaacson, L. M. (1959). Experimental music: composition with an electronic computer.

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, *18*(7), 1527–1554.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Huang, A., & Wu, R. (2016). Deep learning for music. *arXiv preprint arXiv:1606.04930*.

Huang, C.-Z. A., Vaswani, A., Uszkoreit, J., Simon, I., Hawthorne, C., Shazeer, N., ... Eck, D. (2018). Music transformer: Generating music with long-term structure. In *International conference on learning representations.*

Huq, A., Bello, J. P., & Rowe, R. (2010). Automated music emotion recognition: A systematic evaluation. *Journal of New Music Research*, *39*(3), 227-244. Retrieved from https://doi.org/10.1080/09298215.2010.513733 doi: 10.1080/09298215.2010.513733

Huron, D., Anderson, N., & Shanahan, D. (2014). "you can't play a sad song on the banjo:"

acoustic factors in the judgment of instrument capacity to convey sadness. *Empirical Musicology Review*, *9*(1), 29–41.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jaimes, A., Sebe, N., & Gatica-Perez, D. (2006, 01). Human-centered computing: A multimedia perspective. In (p. 855-864). doi: 10.1145/1180639.1180829

Janocha, K., & Czarnecki, W. M. (2017). On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*.

Jaques, N., Gu, S., Turner, R. E., & Eck, D. (2017). Tuning recurrent neural networks with reinforcement learning.

Jensen, K. (2006). Multiple scale music segmentation using rhythm, timbre, and harmony. *EURASIP Journal on Advances in Signal Processing*, *2007*, 073205.

Johnson, D. D. (2017). Generating polyphonic music using tied parallel networks. In *International conference on evolutionary and biologically inspired music and art* (pp. 128–143).

Jordan, M. I., & Jacobs, R. A. (1994). Hierarchical mixtures of experts and the em algorithm. *Neural computation*, *6*(2), 181–214.

Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *International conference on machine learning* (pp. 2342–2350).

Juslin, P. N., & Sloboda, J. A. (2011). Handbook of music and emotion: Theory, research, applications..

Kaiser, F., & Peeters, G. (2013). A simple fusion method of state and sequence segmentation for music structure discovery. In *Proceedings of the 14th international society for music information retrieval conference* (pp. 257–262).

Karpathy, A. (n.d.). *The unreasonable effectiveness of recurrent neural networks.* `https://goo.gl/9rvadx`.

Katharopoulos, A., Vyas, A., Pappas, N., & Fleuret, F. (2020). *Transformers are rnns: Fast autoregressive transformers with linear attention.*

Kim, Y., Schmidt, E., Migneco, R., Morton, B., Scott, J., Speck, J., & Turnbull, D. (2010, 01). Music emotion recognition: A state of the art review. *Proceedings of the 11th International Society for Music Information Retrieval Conference, ISMIR 2010.*

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kong, Q., Li, B., Chen, J., & Wang, Y. (2020). Giantmidi-piano: A large-scale midi dataset for classical piano music. *arXiv preprint arXiv:2010.07061*.

Koory, J. L., & Medley, D. B. (1987). *Management information systems: planning and decision making*. South-Western Thomson Learning.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

Krueger, D., Maharaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N. R., ... Pal, C. (2016). Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*.

Krueger, D., & Memisevic, R. (2015). Regularizing rnns by stabilizing activations. *arXiv preprint arXiv:1511.08400*.

Kumar, A., Irsoy, O., Ondruska, P., Iyyer, M., Bradbury, J., Gulrajani, I., ... Socher, R. (2016). Ask me anything: Dynamic memory networks for natural language processing. In *International conference on machine learning* (pp. 1378–1387).

Kuntsche, E., Mével, L. L., & Berson, I. (2016). Development of the four-dimensional motives for listening to music questionnaire (mlmq) and associations with health and social issues among adolescents. *Psychology of Music*, *44*, 219-233.

Kursa, M. B., Rudnicki, W. R., et al. (2010). Feature selection with the boruta package. *J Stat Softw*, *36*(11), 1–13.

Kurth, F., Müller, M., Damm, D., Fremerey, C., Ribbrock, A., & Clausen, M. (2005, September). Syncplayer - an advanced system for multimodal music access. In *Proceedings of the 6th international conference on music information retrieval* (p. 381-388). Retrieved from `https://doi.org/10.5281/zenodo.1416496` doi: 10.5281/zenodo.1416496

Lamere, P. (2000). *The infinite jukebox.* `http://infinitejukebox.playlistmachinery.com/`. (Accessed: 18-05-2020)

Lancichinetti, A., & Fortunato, S. (2011). Limits of modularity maximization in community detection. *Physical Review E*, *84*(6), 066122.

Lattner, S., Grachten, M., & Widmer, G. (2018). A predictive model for music based on learned interval representations. *arXiv preprint arXiv:1806.08686*.

Laurent, C., Pereyra, G., Brakel, P., Zhang, Y., & Bengio, Y. (2016). Batch normalized recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2016 ieee international conference on* (pp. 2657–2661).

Lavrenko, V., & Pickens, J. (2003). Polyphonic music modeling with random fields. In *Proceedings of the eleventh acm international conference on multimedia* (pp. 120–129).

Le, Q. V., Jaitly, N., & Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*.

LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

Lerdahl, F., & Jackendoff, R. S. (1985). *A generative theory of tonal music*. MIT press.

Lerdahl, F., Jackendoff, R. S., & Jackendoff, R. (1983). *A generative theory of tonal music*. MIT press.

Levitin, D. J. (2006). *This is your brain on music: The science of a human obsession*. Penguin.

Levy, M., & Sandler, M. (2008). Structural segmentation of musical audio by constrained clustering. *IEEE Transactions on Audio, Speech and Language Processing*, *16*(2), 318–326.

Li, J., Tu, Z., Yang, B., Lyu, M. R., & Zhang, T. (2018, October-November). Multi-head attention with disagreement regularization. In *Proceedings of the 2018 conference on empirical methods in natural language processing* (pp. 2897–2903). Brussels, Belgium: Association for Computational Linguistics. Retrieved from `https://www.aclweb.org/anthology/D18-1317` doi: 10.18653/v1/D18-1317

Liang, F., Gotham, M., Johnson, M., & Shotton, J. (2017). Automatic stylistic composition of bach chorales with deep lstm. In *Proceedings of the 18th international society for music information retrieval conference (ismir-17), suzhou, china.*

Livingstone, S. R., Palmer, C., & Schubert, E. (2012). Emotional response to musical repetition. *Emotion*, *12*(3), 552.

Lu, L., Wang, M., & Zhang, H.-J. (2004). Repeating pattern discovery and structure analysis from acoustic music data. In *Proceedings of the 6th acm sigmm international workshop on multimedia information retrieval* (pp. 275–282).

Lyu, Q., Wu, Z., & Zhu, J. (2015). Polyphonic music modelling with lstm-rtrbm. In *Proceedings of the 23rd acm international conference on multimedia* (pp. 991–994).

MacDorman, K. F., Stuart Ough Chin-Chang Ho. (2007). Automatic emotion prediction of song excerpts: Index construction, algorithm design, and empirical comparison. *Journal of New Music Research*, *36*(4), 281–299.

Madhok, R., Goel, S., & Garg, S. (2018). Sentimozart: Music generation based on emotions. In *Icaart.*

Mann, Y. (2016). Ai duet. *Experiments with Google. See, https://experiments. withgoogle. com/ai/ai-due t*.

Marcus, M. P., Marcinkiewicz, M. A., & Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, *19*(2), 313–330.

Martens, J., & Sutskever, I. (2011). Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th international conference on machine learning (icml-11)* (pp. 1033–1040).

Martin, C. P., Ellefsen, K. O., & Torresen, J. (2018). Deep predictive models in interactive music. *arXiv preprint arXiv:1801.10492*.

Mcadams, S., Vines, B., Vieillard, S., Smith, B., & Reynolds, R. (2004, 12). Influences of large-scale form on continuous ratings in response to a contemporary piece in a live concert setting. *Music Perception*, *22*, 297-350. doi: 10.1525/mp.2004.22.2.297

McFee, B., & Ellis, D. (2014a). Analyzing song structure with spectral clustering. In *Proceedings of the 15th international society for music information retrieval conference* (pp. 405–410).

McFee, B., & Ellis, D. P. W. (2014b). Learning to segment songs with ordinal linear discriminant analysis. *Self*, *275*, 330.

McFee, B., & Kinnaird, K. M. (2019). Improving structure evaluation through automatic hierarchy expansion. In *Proceedings of the 20th international society for music information retrieval conference.*

McFee, B., Lostanlen, V., McVicar, M., Metsai, A., Balke, S., Thomé, C., ... Weiss, A. (2020, January). *librosa/librosa: 0.7.2.* Zenodo. Retrieved from `https://doi.org/10.5281/zenodo.3606573` doi: 10.5281/zenodo.3606573

McFee, B., Lostanlen, V., Metsai, A., McVicar, M., Balke, S., Thomé, C., ... Kim, T. (2020, July). *librosa/librosa: 0.8.0.* Zenodo. Retrieved from `https://doi.org/10.5281/zenodo.3955228` doi: 10.5281/zenodo.3955228

McFee, B., Nieto, O., Farbood, M. M., & Bello, J. P. (2017). Evaluating hierarchical structure in music annotations. *Frontiers in Psychology*, *8*, 1337.

McFee, B., Raffel, C., Liang, D., Ellis, D. P. W., McVicar, M., Battenberg, E., & Nieto, O. (2015). librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference* (pp. 18–25).

Mehr, S. A., Singh, M., Knox, D., Ketter, D. M., Pickens-Jones, D., Atwood, S., ... others (2019). Universality and diversity in human song. *Science*, *366*(6468).

Melis, G., Dyer, C., & Blunsom, P. (2017). On the state of the art of evaluation in neural language models. *CoRR*, *abs/1707.05589*. Retrieved from `http://arxiv.org/abs/1707.05589`

Merity, S., Xiong, C., Bradbury, J., & Socher, R. (2016). Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Meyer, L. B. (2008). *Emotion and meaning in music*. University of chicago Press.

Mihelač, L., & Povh, J. (2020). The impact of the complexity of harmony on the acceptability of music. *ACM Transactions on Applied Perception (TAP)*, *17*(1), 1–27.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, *63*(2), 81.

Mion, L., & De Poli, G. (2008). Score-independent audio features for description of music

expression. *IEEE Transactions on Audio, Speech, and Language Processing*, *16*(2), 458–466.

Mirikitani, D. T., & Nikolaev, N. (2010). Recursive bayesian recurrent neural networks for time-series modeling. *IEEE Transactions on Neural Networks*, *21*(2), 262–274.

Moon, T., Choi, H., Lee, H., & Song, I. (2015). Rnndrop: A novel dropout for rnns in asr. In *Automatic speech recognition and understanding (asru), 2015 ieee workshop on* (pp. 65–70).

Mourchid, Y., El Hassouni, M., & Cherifi, H. (2016). Image segmentation based on community detection approach. *International Journal of Computer Information Systems and Industrial Management Applications*, 2150–7988.

Mozer, M. C. (1994). Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, *6*(2-3), 247–280.

Müller, M. (2015). *Fundamentals of music processing: Audio, analysis, algorithms, applications*. Springer.

Müller, M., Chew, E., & Bello, J. P. (2016). Computational music structure analysis (Dagstuhl seminar 16092). In *Dagstuhl reports* (Vol. 6).

Müller, M., & Kurth, F. (2006). Towards structural analysis of audio recordings in the presence of musical variations. *EURASIP Journal on Advances in Signal Processing*, *2007*(1), 089686.

Neelakantan, A., Le, Q. V., & Sutskever, I. (2015). Neural programmer: Inducing latent programs with gradient descent. *arXiv preprint arXiv:1511.04834*.

Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady an ussr* (Vol. 269, pp. 543–547).

Newman, M. E. (2004a). Analysis of weighted networks. *Physical Review E*, *70*(5), 056131.

Newman, M. E. (2004b). Fast algorithm for detecting community structure in networks. *Physical Review E*, *69*(6), 066133.

Nieto, O., & Bello, J. P. (2014). Music segment similarity using 2d-Fourier magnitude coefficients. In *Ieee international conference on acoustics, speech and signal processing* (pp. 664–668).

Nieto, O., & Bello, J. P. (2015). MSAF: Music structure analysis framework. In *Proceedings of the 16th international society for music information retrieval conference.*

Oberauer, K. (2019). Working memory and attention–a conceptual analysis and review. *Journal of cognition*, *2*(1).

Olah, C. (n.d.). *Understanding lstm networks.* `colah.github.io/posts/2015-08-Understanding-LSTMs/`.

Olah, C., & Carter, S. (2016). Attention and augmented recurrent neural networks. *Distill*. Retrieved from http://distill.pub/2016/augmented-rnns doi: 10.23915/distill.00001

Papadopoulos, A., Roy, P., & Pachet, F. (2016). Assisted lead sheet composition using flowcomposer. In *International conference on principles and practice of constraint programming* (pp. 769–785).

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., ... Fernández, R. (2016). The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*.

Pascanu, R., Gulcehre, C., Cho, K., & Bengio, Y. (2013). How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Patel, A. D. (2003). Language, music, syntax and the brain. *Nature neuroscience*, *6*(7), 674–681.

Paulus, J. (2010). Improving Markov model based music piece structure labelling with acoustic information. In *Proceedings of the 11th international society for music information retrieval conference* (pp. 303–308).

Paulus, J., & Klapuri, A. (2009). Music structure analysis using a probabilistic fitness measure and a greedy search algorithm. *IEEE Transactions on Audio, Speech, and Language Processing*, *17*(6), 1159–1170.

Paulus, J., Müller, M., & Klapuri, A. (2010, August). State of the art report: Audio-based music structure analysis. In *Proceedings of the 11th international society for music information retrieval conference* (p. 625-636). Retrieved from https://doi.org/10.5281/zenodo.1417289 doi: 10.5281/zenodo.1417289

Pease, A., & Colton, S. (2011). On impact and evaluation in computational creativity: A discussion of the turing test and an alternative proposal. In *Proceedings of the aisb symposium on ai and philosophy* (Vol. 39).

Pease, A., Mahmoodi, K., & West, B. J. (2018). Complexity measures of music. *Chaos, Solitons & Fractals*, *108*, 82–86.

Peeters, G. (2003). Deriving musical structures from signal analysis for music audio summary generation: "sequence" and "state" approach. In *International symposium on computer music modeling and retrieval* (pp. 143–166).

Pincus, S. M. (1991). Approximate entropy as a measure of system complexity. *Proceedings of the National Academy of Sciences*, *88*(6), 2297–2301.

Posner, J., Russell, J. A., & Peterson, B. S. (2005). The circumplex model of affect: An integrative approach to affective neuroscience, cognitive development, and psychopathology. *Development and psychopathology*, *17*(3), 715–734.

Pujol, J. M., Béjar, J., & Delgado, J. (2006). Clustering algorithm for determining community structure in large networks. *Physical Review E*, *74*(1), 016107.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural networks*, *12*(1), 145–151.

Raffel, C. (2016). *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. Columbia University.

Raffel, C., & Ellis, D. P. (2014). Intuitive analysis, creation and manipulation of midi data with pretty midi. In *15th international society for music information retrieval conference late breaking and demo papers* (pp. 84–93).

Richman, J. S., & Moorman, J. R. (2000). Physiological time-series analysis using approximate entropy and sample entropy. *American Journal of Physiology-Heart and Circulatory Physiology*, *278*(6), H2039–H2049.

Roland, P. (2002). The music encoding initiative (mei). In *Proceedings of the first international conference on musical applications using xml* (Vol. 1060, pp. 55–59).

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Saari, P., Eerola, T., & Lartillot, O. (2010). Generalizability and simplicity as criteria in feature selection: Application to mood classification in music. *IEEE Transactions on Audio, Speech, and Language Processing*, *19*(6), 1802–1812.

Schacter, D., Gilbert, D. T., & Wegner, D. M. (2011). *Psychology (2nd edition)*. New York: Worth.

Schaffrath, H., & Huron, D. (1995). The essen folksong collection in the humdrum kern format. *Menlo Park, CA: Center for Computer Assisted Research in the Humanities*.

Schmidt, E. M., & Kim, Y. E. (2010). Prediction of time-varying musical mood distributions from audio. In *Ismir* (pp. 465–470).

Schmidt, E. M., Turnbull, D., & Kim, Y. E. (2010). Feature selection for content-based, time-varying musical emotion regression. In *Proceedings of the international conference on multimedia information retrieval* (pp. 267–274).

Schuller, B., Steidl, S., Batliner, A., Vinciarelli, A., Scherer, K., Ringeval, F., ... others (2013). The interspeech 2013 computational paralinguistics challenge: Social signals, conflict,

emotion, autism. In *Proceedings interspeech 2013, 14th annual conference of the international speech communication association, lyon, france.*

Schäfer, T., Sedlmeier, P., Städtler, C., & Huron, D. (2013). The psychological functions of music listening. *Frontiers in Psychology*, *4*, 511. Retrieved from https://www.frontiersin.org/article/10.3389/fpsyg.2013.00511 doi: 10.3389/fpsyg.2013.00511

Serrà, J., Zanin, M., Herrera, P., & Serra, X. (2012). Characterization and exploitation of community structure in cover song networks. *Pattern Recognition Letters*, *33*(9), 1032–1041.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell system technical journal*, *27*(3), 379–423.

Shepard, R., & Levitin, D. (2002). Cognitive psychology and music. *Foundations of cognitive psychology: Core readings*, 503–514.

Shin, A., Crestel, L., Kato, H., Saito, K., Ohnishi, K., Yamaguchi, M., ... Harada, T. (2017). Melody generation for pop music via word representation of musical properties. *arXiv preprint arXiv:1710.11549*.

Siegelmann, H. T., & Sontag, E. D. (1995). On the computational power of neural nets. *Journal of computer and system sciences*, *50*(1), 132–150.

Sigtia, S., Benetos, E., & Dixon, S. (2016). An end-to-end neural network for polyphonic piano music transcription. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *24*(5), 927–939.

Simon, I., Roberts, A., Raffel, C., Engel, J., Hawthorne, C., & Eck, D. (2018). Learning a latent space of multitrack measures. *arXiv preprint arXiv:1806.00195*.

Sloboda, J. A. (1991). Music structure and emotional response: Some empirical findings. *Psychology of music*, *19*(2), 110–120.

Smith, J., Kawasaki, Y., & Goto, M. (2019, November). Unmixer: An interface for extracting and remixing loops. In *Proceedings of the 20th international society for music information retrieval conference* (p. 824-831). Retrieved from https://doi.org/10.5281/zenodo.3527938 doi: 10.5281/zenodo.3527938

Smith, J. B., & Chew, E. (2013). Using quadratic programming to estimate feature relevance in structural analyses of music. In *Proceedings of the 21st acm international conference on multimedia* (pp. 113–122).

Smith, J. B. L., Burgoyne, J. A., Fujinaga, I., De Roure, D., & Downie, J. S. (2011, October). Design and creation of a large-scale database of structural annotations. In *Proceedings of the 12th international society for music information retrieval conference* (p. 555-560). Retrieved from https://doi.org/10.5281/zenodo.1416884 doi: 10.5281/zenodo

.1416884

Song, Y., & Simon, D. (2015). How well can a music emotion recognition system predict the emotional responses of participants? In *Sound and music computing conference (smc)* (pp. 387–392).

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.

Stöter, F.-R., Liutkus, A., & Ito, N. (2018). The 2018 signal separation evaluation campaign. In *International conference on latent variable analysis and signal separation* (pp. 293–305).

Sturm, B., Santos, J. F., & Korshunova, I. (2015). Folk music style modelling by recurrent neural networks with long short term memory units. In *16th international society for music information retrieval conference.*

Sturm, B. L., & Ben-Tal, O. (2017). Taking the models back to music practice: evaluating generative transcription models built using deep learning. *Journal of Creative Music Systems*, *2*(1).

Sturm, B. L., Iglesias, M., Ben-Tal, O., Miron, M., & Gómez, E. (2019). Artificial intelligence and music: open questions of copyright law and engineering praxis. In *Arts* (Vol. 8, p. 115).

Sturm, B. L., Santos, J. F., Ben-Tal, O., & Korshunova, I. (2016). Music transcription modelling and composition using deep learning. *arXiv preprint arXiv:1604.08723*.

Sukhbaatar, S., Weston, J., Fergus, R., Szlam, A., Weston, J., & Fergus, R. (2015). End-to-end memory networks. *Nips*, 1–11. Retrieved from `https://arxiv.org/pdf/1503.08895.pdfhttp://arxiv.org/abs/1503.08895` doi: v5

Sutskever, I. (2013). Training recurrent neural networks.

Sutskever, I., Hinton, G. E., & Taylor, G. W. (2009). The recurrent temporal restricted boltzmann machine. In *Advances in neural information processing systems* (pp. 1601–1608).

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International conference on machine learning* (pp. 1139–1147).

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (pp. 3104–3112).

Temperley, D. (2004). *The cognition of basic musical structures*. MIT press.

Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, *4*(2), 26–31.

Todd, P. M. (1989). A connectionist approach to algorithmic composition. *Computer Music Journal*, *13*(4), 27–43.

Toussaint, G. T., & Trochidis, K. (2018). On measuring the complexity of musical rhythm. In *2018 9th ieee annual ubiquitous computing, electronics & mobile communication conference (uemcon)* (pp. 753–757).

Uria, B., Côté, M.-A., Gregor, K., Murray, I., & Larochelle, H. (2016). Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, *17*(1), 7184–7220.

Van den Oord, A., Dieleman, S., & Schrauwen, B. (2013). Deep content-based music recommendation. In *Advances in neural information processing systems* (pp. 2643–2651).

van Goethem, A., & Sloboda, J. (2011). The functions of music for affect regulation. *Musicae Scientiae*, *15*(2), 208-228. Retrieved from https://doi.org/10.1177/1029864911401174 doi: 10.1177/1029864911401174

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 6000–6010).

Veltkamp, R. C., Wiering, F., & Typke, R. (2008). Content based music retrieval. In *Encyclopedia of multimedia* (pp. 97–98). Springer.

Vohra, R., Goel, K., & Sahoo, J. (2015). Modeling temporal dependencies in data using a dbn-lstm. In *Data science and advanced analytics (dsaa), 2015. 36678 2015. ieee international conference on* (pp. 1–4).

Waite, E. (2016). *Generating long-term structure in songs and stories.* Retrieved 2020-08-04, from https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn/

Walder, C. (2016). Modelling symbolic music: Beyond the piano roll. In *Asian conference on machine learning* (pp. 174–189).

Wattenberg, M. (2000). *The shape of song.* http://turbulence.org/Works/song/method/method.html. (Accessed: 18-05-2020)

Weiss, C., & Müller, M. (2015). Tonal complexity features for style classification of classical music. In *2015 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 688–692).

Weston, J., Chopra, S., & Bordes, A. (2014). Memory networks. *arXiv preprint arXiv:1410.3916*.

White, J. D. (1984). *The analysis of music*. Metuchen, NJ: Scarecrow Press.

Widmer, G., & Goebl, W. (2004). Computational models of expressive music performance: The state of the art. *Journal of New Music Research*, *33*(3), 203–216.

Williams, R. J. (1992). Training recurrent networks using the extended kalman filter. In *Neural*

networks, 1992. ijcnn., international joint conference on (Vol. 4, pp. 241–246).

Wöllner, C. (2013). How to quantify individuality in music performance? *Frontiers in psychology*, *4*, 361.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning* (pp. 2048–2057).

Xu, M., Li, X., Xianyu, H., Tian, J., Meng, F., & Chen, W. (2015). Multi-scale approaches to the mediaeval 2015 "emotion in music" task. In *Mediaeval.*

Yang, L.-C., & Lerch, A. (2020). On the evaluation of generative models in music. *Neural Computing and Applications*, *32*(9), 4773–4784.

Yang, X., Dong, Y., & Li, J. (2017). Review of data features-based music emotion recognition methods. *Multimedia Systems*, *24*, 365-389.

Yang, Y.-H., & Chen, H. H. (2012). Machine recognition of music emotion: A review. *ACM Trans. Intell. Syst. Technol.*, *3*, 40:1-40:30.

Ycart, A., Benetos, E., et al. (2017). A study on lstm networks for polyphonic music sequence modelling..

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Zentner, M., Grandjean, D., & Scherer, K. (2008, 09). Emotions evoked by the sound of music: Characterization, classification, and measurement. *Emotion (Washington, D.C.)*, *8*, 494-521. doi: 10.1037/1528-3542.8.4.494

Zhang, J., Xianglin, H., Yang, L., & Nie, L. (2016, 06). Bridge the semantic gap between pop music acoustic feature and emotion: build an interpretable model. *Neurocomputing*, *208*. doi: 10.1016/j.neucom.2016.01.099

Zhang, K., Zhang, H., Li, S., Yang, C., & Sun, L. (2018). The pmemo dataset for music emotion recognition. In *Proceedings of the 2018 acm on international conference on multimedia retrieval* (p. 135–142). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/3206025.3206037` doi: 10.1145/3206025.3206037

Zhang, S., Wu, Y., Che, T., Lin, Z., Memisevic, R., Salakhutdinov, R. R., & Bengio, Y. (2016). Architectural complexity measures of recurrent neural networks. In *Advances in neural information processing systems* (pp. 1822–1830).

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *67*(2), 301–320.