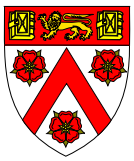
UNIVERSITY OF
CAMBRIDGE

The resurgence of structure in deep neural networks

Petar Veličković



Trinity College

This dissertation is submitted on 15 January 2019
for the degree of Doctor of Philosophy

The resurgence of structure in deep neural networks

Petar Veličković

Abstract

Machine learning with deep neural networks (*"deep learning"*) allows for learning complex features directly from raw input data, completely eliminating hand-crafted, "hard-coded" feature extraction from the learning pipeline. This has lead to state-of-the-art performance being achieved across several—previously disconnected—problem domains, including computer vision, natural language processing, reinforcement learning and generative modelling. These success stories nearly universally go hand-in-hand with availability of *immense* quantities of labelled training examples (*"big data"*) exhibiting simple *grid-like* structure (e.g. text or images), exploitable through convolutional or recurrent layers. This is due to the extremely large number of degrees-of-freedom in neural networks, leaving their generalisation ability vulnerable to effects such as overfitting.

However, there remain many domains where extensive data gathering is not always appropriate, affordable, or even feasible. Furthermore, data is generally organised in more complicated kinds of structure—which most existing approaches would simply discard. Examples of such tasks are abundant in the *biomedical* space; with e.g. small numbers of subjects available for any given clinical study, or relationships between proteins specified via interaction networks. I hypothesise that, if deep learning is to reach its full potential in such environments, we need to reconsider "hard-coded" approaches—integrating assumptions about inherent structure in the input data directly into our architectures and learning algorithms, through *structural inductive biases*. In this dissertation, I directly validate this hypothesis by developing three *structure-infused neural network* architectures (operating on sparse multimodal and graph-structured data), and a *structure-informed learning algorithm* for graph neural networks, demonstrating significant outperformance of conventional baseline models and algorithms.

Declaration

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. It is not substantially the same as any that I have submitted, or am concurrently submitting, for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my dissertation has already been submitted, or is being concurrently submitted, for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. This dissertation does not exceed the prescribed limit of 60 000 words.

Petar Veličković
15 January 2019

Acknowledgements

Finally completing this dissertation brings to an end a truly extraordinary three-year period of my life. While it has challenged my sanity in ways that I could never have been able to foresee, it has also given me the opportunity to have some of the best experiences of my life. I truly believe that I have come out of the entire experience as, primarily, a better person. However, this would have not been even remotely possible without support of some of the most amazing and generous people in this world. They have simultaneously celebrated my successes and made sure to always remind me that my work has value, whenever times were hard. Here, I will humbly try to give them due credit.

I will start by acknowledging my exceptional *parents*, **Alma** and **Dušan Veličković**. They have helped me see through this academic journey with unconditional love, and understanding for the divergence between our professional worlds. As through all of my life, they have truly provided the backbone to my efforts. I hope that I have managed to make them proud remotely as much as they've made my life fulfilling.

This PhD wouldn't be remotely as fruitful without the endless support of my *supervisor*, **Pietro Liò**. Ever since the "early" days in the Computer Science Tripos, Pietro put the utmost of faith in me, and fully rightly taught me that a PhD is all about *expanding the network*, accommodating my every research-related whim. I'm hopeful that I have justified the confidence I've received from him, and remain eternally thankful.

Besides Pietro, over the course of this PhD I was very lucky to interact with **Nicholas Lane** and **Yoshua Bengio**, both of whom put extreme faith in me when I did not have any way of substantiating this faith. The work I've done in collaboration with them (at *Nokia Bell Labs* and *Mila*, respectively) forms the essence of this dissertation. I cannot thank them enough for the wonderful opportunities. This also provided essential credentials for my next pursuit (at *DeepMind*), and I would like to particularly thank **Raia Hadsell** for her continued interest in my work, and look forward to working with her for years to come. Credits are also due to **Jovana Mitrović** and **Irina Higgins**, both of whom played a key supportive role in guiding me through the process of becoming a Research Scientist.

When it comes to my formation as a machine learning researcher, nobody deserves more credit than **Adriana Romero** and **Devon Hjelm**, direct supervisors of my work at Mila. Adriana has taught me all the tricks of the trade; the hidden methodological and intuitive details that research papers scarcely show. Devon, on the other hand, truly convinced me that impossible is nothing, with his endlessly persistent and driven mentorship. Both of them, being extremely hands-on, managed to extract the utmost of my potential, and I am grateful to have been able to get a chance to work with them.

In these three years, I had a chance to collaborate with some brilliant researchers. Ones that have impacted my formation the most—either through very instructive insights or immense collegial spirit—are **Duo Wang, Sourav Bhattacharya, Pietro Sormanni, Guillem Cucurull, Arantxa Casanova, Thomas Kipf, Jian Tang, William Fedus** and **William Hamilton**—whom I also give exceptional thanks to for inviting me to co-organise a workshop on graph representation learning, which has proven to be a truly invaluable experience.

Perhaps no experience has been more humbling than getting to *supervise* students' project work. Even more so, seeing that work published at various strong venues, and seeing many of the people I advised go on to building successful careers in research of their own. In this respect, I'd like to acknowledge **Benjamin Day, Ioana Bica, Momchil Peychev, Tudor Ţiplea, Edgar Liberis, Laurynas Karazija** and **Nathaniel McAleese-Park**. Foremost, they are all exceptional people with immense potential, and I hope that I managed to justify their faith by playing a useful role in shaping their interests.

Outside research, some of the most computer science-related fun I've had these three years involved participating in *competitive programming* and being a *machine learning educator*. In the first aspect, I thank **Dimitrije Erdeljan, Marko Stanković** and **Dušan Živanović** for being truly the best team of competitive programmers one can work with. In the second, I cannot even begin to explain how important my collaboration with *Cambridge Spark* was—in terms of substantially improving my stage confidence as well as meeting some truly exceptional people. As such, I thank **Annalisa Occhipinti** for giving me the initial opportunity for collaborating with this company, and the CEO, **Raoul-Gabriel Urma**, for entrusting me with so many additional engagements since then. Of course, thanks are also due to all the amazing people at Cambridge Spark I worked with throughout my PhD: **Thibaut Lienart, Valentin Dalibard, Olivia Hughes, Elena Hatzimihali, Ekaterina Kochmar, Chih-Chun Chen** and **Ryan-Rhys Griffiths**.

I was exceptionally lucky to perform a research visit to Montréal's Mila *two* times during my PhD. Therein, I have met extraordinary machine learning researchers, but primarily wonderful and highly welcoming people. Highlighting only a few is very hard—but I'd like to thank **Rosemary Ke**, **Chiheb Trabelsi**, **Joseph Paul Cohen**, **Assya Trofimov**, **Dmitry Serdyuk**, **Dendi Suhubdy**, **Philippe Lacaille**, **Samuel Lavoie-Marchildon**, **Yikang Shen**, **Konstantinos Drossos**, **João Felipe Santos**, **Christopher Beckham**, **Olexa Bilaniuk**, **Philemon Brakel**, **François Corneau-Tremblay** and **Alex Fedorov** for making my experience a truly unforgettable one. But, utmostly, I'd love to thank **Amina Madzhun**, **Jae Hyun Lim**, **Shawn Tan** and **Chin-Wei Huang**, who accompanied me to late night hours in the 3248 lab during the most pressuring deadline push of my life. Without their overwhelming support, this dissertation would likely have missed its last chapter.

Throughout the development of this dissertation I always had the support from a select group of really close people, who—in many respects—were like a second family to me: **Nikola Jovanović**, **Cătălina Cangea**, **Miloš Stanojević**, **Ana Šemrov**, **Vanja Šarković**, **Djordje Nikolić** and **Marina Ivanović**. Enumerating all the ways in which these people were invaluable to my progress would probably occupy more pages than this dissertation—and therefore, here I would just like to thank them for all the times they were there for me. It will never be forgotten.

To **Emma Rocheteau**, I would like to say that she is truly exceptional. She has exceeded the limits of human kindness I thought were possible. At least when it comes to making sure that I am in the best possible state, she is ready to go to truly extreme lengths—even when things are critical with her as well. I hope that only the best things happen to her from now on, as she most certainly deserves it. She has certainly earned my endless respect and eternal friendship. **Merci beaucoup!**

Last, but certainly not least, I would like to express indefinite gratitude to **Andreea-Ioana Deac**. If there is one person that consistently reinforced my (debatable) self-confidence, and made sure that I was on track, both academically and psychologically—often sacrificing countless hours of her own time to do so—that would definitely be her. From the very fruitful research collaborations to all the random and wonderful *bonaventures* that followed, she has been a really lovely person to have around, and never failed to remind me that what I'm doing is worthwhile. I am truly lucky to have such a caring person in my life, and it is no understatement that completing this dissertation would not have been possible without her never-ending support. **Mulțumesc foarte mult!**

Contents

Chapter 1

Introduction

> "Language makes *infinite* use
> of *finite* media."
>
> Wilhelm von Humboldt

The initial success stories of computer science (primarily related to areas such as algorithm design and analysis, computer architecture or programming languages) revolved around efficiently breaking down problems into simpler problems, in such a way that the steps to solve them may eventually be specified unambiguously, as a sequence of "hard-coded" instructions that a computer is capable of executing. This approach was pervasive in the historical work on artificial intelligence as well—aiming to represent knowledge about the world as a *knowledge base* of axioms, along with simple *rules* for deriving new knowledge [123].

Despite its initial successes, it quickly became evident that the symbolic approach to artificial intelligence becomes intractable in many problems of interest. The algorithms are required to contain an exponentially growing space of intermediate conclusions—many of which irrelevant or trivially symmetric to each other—to reach a final decision.

More importantly, it turned out that for many problems of interest it is *extremely hard* to manually specify rigorous axioms and rules that a computer could use for making conclusions. This is sometimes referred to as the *symbol grounding problem* [66]: relying entirely on hand-crafted mathematical elements (such as constants, functions and predicates) can hardly be reconciled with the richness of signal in the real world, and therefore these methods typically lack robustness to minor perturbations of the input problem setup. As a guiding example, I will leverage a standard computer vision task: glyph recognition on the notMNIST [19] dataset (Figure 1.1).

Figure 1.1: The notMNIST dataset of glyphs corresponding to characters A–J. For humans, the associated classification task—of predicting the character depicted in the glyph—is trivial to perform, but near-impossible to conceptualise algorithmically.

The notMNIST dataset, designed as a step-up in complexity from the handwritten digit recognition of MNIST [103], is concerned with predicting the character depicted in a given input glyph (represented as a 28×28 grayscale image; i.e., matrix of pixel intensities). For humans, this task is often *trivial*; the majority of the glyphs shown in Figure 1.1 may quickly be recognised as the character A. However, formulating a rigorous set of classification rules turns out to be quite difficult. This difficulty stems primarily from the *variability* of real-world signals: the large number of ways (*fonts*) to effectively encode the "substance" of an A is not readily amenable to a rule-based system that looks at individual pixels. Even simple transformations, such as shifts, scales and rotations may completely fool such a system without compromising human performance[1].

Unlike such a rule-based classifier, humans have been *exposed* to a significant number of A characters during their lifetimes, and have therefore *learnt* the complex features that make up an A, without ever needing to rigorously define a set of rules for doing so. **Machine learning**, at its core, aims to capture exactly this—systems capable of *generalising from past experiences*. It has, therefore, become the dominant approach to artificial intelligence in recent years; especially through *deep neural networks* (commonly known as **deep learning** [101] in this context).

[1] It should be noted, of course, that modern methods based on deep neural networks are not immune to variants of this issue either—in this space often referred to as *adversarial attacks* [168]. These, however, often produce fooling inputs that substantially escape the *true data distribution*.

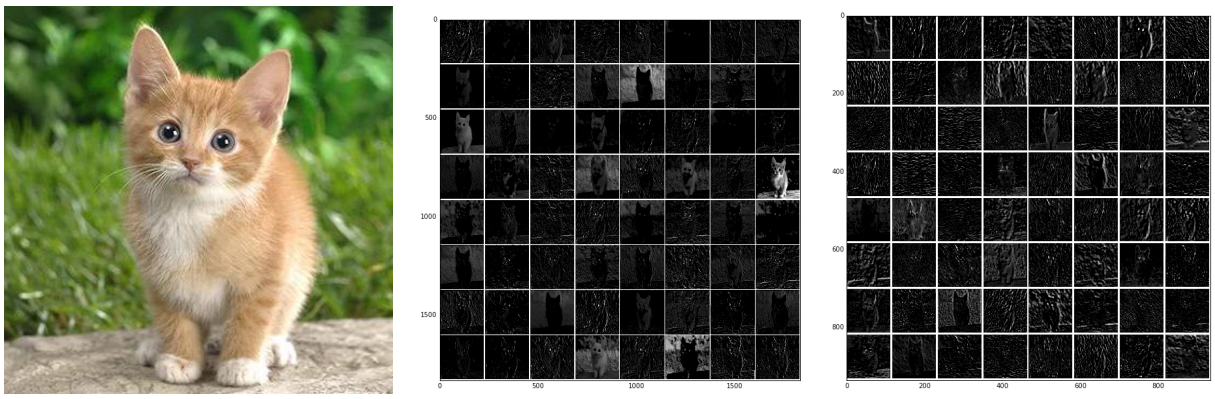


Figure 1.2: **Left:** An input image (of the *tabby cat* class). A deep neural network (VGG-16 [160]) pre-trained on ImageNet is capable of extracting meaningful representations in both its shallow (**middle**) and deep (**right**) layers. The shallower layers typically extract rudimentary features (such as separating the foreground from the background) while, at the deeper stages of processing, the network highlights distinct "cat-like" properties, such as its eyes, ears and fur.

1.1 The successes and pitfalls of deep learning

Deep neural networks (DNNs) tackle the problem from a *representation learning* [11] perspective; they are composed of a stack of *feature-extracting layers*, with the first layer processing *raw inputs*, and each subsequent layer receiving the output of the previous layer. As such, DNNs build up a gradually more complex *representation* of the input and eliminate the need for hand-crafted feature extraction. Given a sufficiently large *training set* of input/output examples, the network iteratively adjusts its parameters (typically using a first-order method [143] such as gradient descent) in order to optimise the error of the network's predictions on those examples compared to the ground-truth output. As a consequence, the network gradually acquires better representations of the data as well, with the "shallower" layers specialising to detect simple input features (such as edges and contours in an image), with "deeper" layers detecting more complex clues. Figure 1.2 provides a visualisation of this phenomenon in the case of *image classification* on the ImageNet dataset [144] (where it is typically easiest to conceptualise).

This method assumes a direction which is in stark contrast to the symbolic approaches; *minimal* assumptions about the specifics of the task are given, relying on the model to automatically infer them through observing the training data. In conjunction with specialised architectures that use *parameter sharing* to exploit simple redundancies and invariances in *grid-structured* data—such as *convolutional neural networks* (CNNs) [102] for images and *recurrent neural networks* (RNNs) [41, 74, 82] for sequences—these techniques currently hold the state-of-the-art result on many challenging tasks of interest. Their resurgence in the 21st century may be primarily attributed to the ability to har-

ness large quantities of training data (the *"big data"* phenomenon) and an upturn in specialised computer architectures for parallel processing (primarily graphics processing units (GPUs), with fully-dedicated architectures such as the tensor processing unit (TPU) [83] being proposed recently).

Under tasks with a large dataset and simplistic structural invariances, deep neural networks typically dominate. The scope of potential applications is extremely varied, with state-of-the-art (and often *superhuman*) performance being attained across areas such as *computer vision* [98], *speech recognition* [70], *natural language processing* [166], *reinforcement learning* [120], *game-playing* [158] and *generative modelling* [54]. What is particularly impressive is that the *same* kind of architecture may often be applied in wildly different scenarios, achieving competitive performance on all of them. This allowed, for the first time, the seamless connection between previously disjoint disciplines of computer science. Historically, tackling each problem in these domains required extensive input from experts for creation of appropriate hand-crafted features. Deep neural networks allow for automatic inference of useful features, purely through observing large quantities of training data, therefore often *eliminating* the domain expert from the pipeline.

While the successes of deep neural networks are indisputable and certainly correspond to the most promising direction of artificial intelligence in recent times, they are far from a catch-all solution to everything. Even if we assume that the networks do not have to perform well in the presence of *adversarial inputs* [55]; a phenomenon already known to leave most deep architectures particularly vulnerable[2], issues arise as soon as we step out of the "big data" environment and/or are forced to deal with more general kinds of input structure (such as *graph-structured data* or data lying on *manifolds*).

As modern-day neural networks often have parameter counts in the *millions*, they will tend to easily *overfit* to their inputs on a small training set—constructing representations that *memorise their training data* rather than generalise to unseen inputs (which is the essential property of successful machine learning systems!). Furthermore, applying neural networks to complex-structured inputs makes it substantially less trivial to share parameters in ways that are simultaneously *meaningful* (do not lose representational power) and *scalable* (computationally efficient, and ideally deployable on GPUs). The most common response to such scenarios is to *gather more data* (nowadays possible even outside the industrial environment, e.g. by leveraging tools such as the Amazon Mechanical Turk [18] to effectively harness human annotators) and to *discard the additional structure*.

[2] While tangential to the aims of the work presented in this dissertation, I acknowledge *defence against adversarial examples* as a very important research direction.

Both of these paradigms work only to an extent—in many cases, discarding the structure can cause severe losses in performance (as is the case with many *node classification* tasks on graphs [92, 187]). Similarly, gathering more data is not always *appropriate*, *affordable* or even *feasible*:

- In some cases, we may not know upfront what the task of interest is—and therefore not be able to determine appropriate *labels* (ground-truth outputs) for the data;
- For some tasks, data can be reliably collected only by *trained professionals*; a simple example of this is medical image segmentation, where every pixel of the input needs to be carefully labelled by an expert. In this setting, techniques such as *active learning* [46] may be applied to carefully choose which examples are the most important to be labelled.
- If we are aiming to detect an extremely *rare* event, such as *diagnosing a rare genetic disease*, *any* training dataset is fundamentally limited by the number of examples possessing this characteristic, and extending it further is impossible[3].

1.2 Reintroducing *structural* inductive biases

The contributions I will present in this dissertation are primarily concerned with alleviating the issues outlined above in the setting where there is additional *structure* in the data which may be exploited. A common way in which additional knowledge about data may be exploited is by applying an appropriate **inductive bias** to the model.

Typically, given a particular (machine) learning setup, one may find a *space* of possible solutions (e.g. parameter assignments) to the learning problem that exhibit equally "good" performance [57]. *Inductive biases* [118], broadly speaking, encourage the learning algorithm to prioritise solutions with certain properties. While there are many ways to encode such biases—e.g. explicit regularisation objectives [115] or choices of prior distributions in a Bayesian model [60]—I have restricted my attention specifically to methods that incorporate structural assumptions directly into the learning architecture or algorithm. This can be seen as a "meet-in-the-middle" approach, combining aspects of classical symbolic artificial intelligence with modern deep architectures for representation learning.

[3] With the caveat that *data augmentation* through generative models such as variational autoencoders (VAEs) [89] or generative adversarial networks (GANs) [54] could provide a way to artificially expand such datasets [199]—albeit substantial further research is necessary in order to verify the viability of such examples in *safety-critical* domains such as biomedicine.

By directly encoding the *structural* inductive biases present in data, I have recovered models that are more *data-efficient*, enabling a leap in predictive power—especially on smaller training datasets. While the specific research questions I have tackled through my contributions will be thoroughly outlined in the next section, I would like to highlight that this is by no means an isolated effort, and in fact represents a major recent push within the machine learning community—as the amount of "low hanging fruit" tasks for deep neural networks deteriorated over the past years. For reasons of space, I will only highlight a few such directions and architectures here:

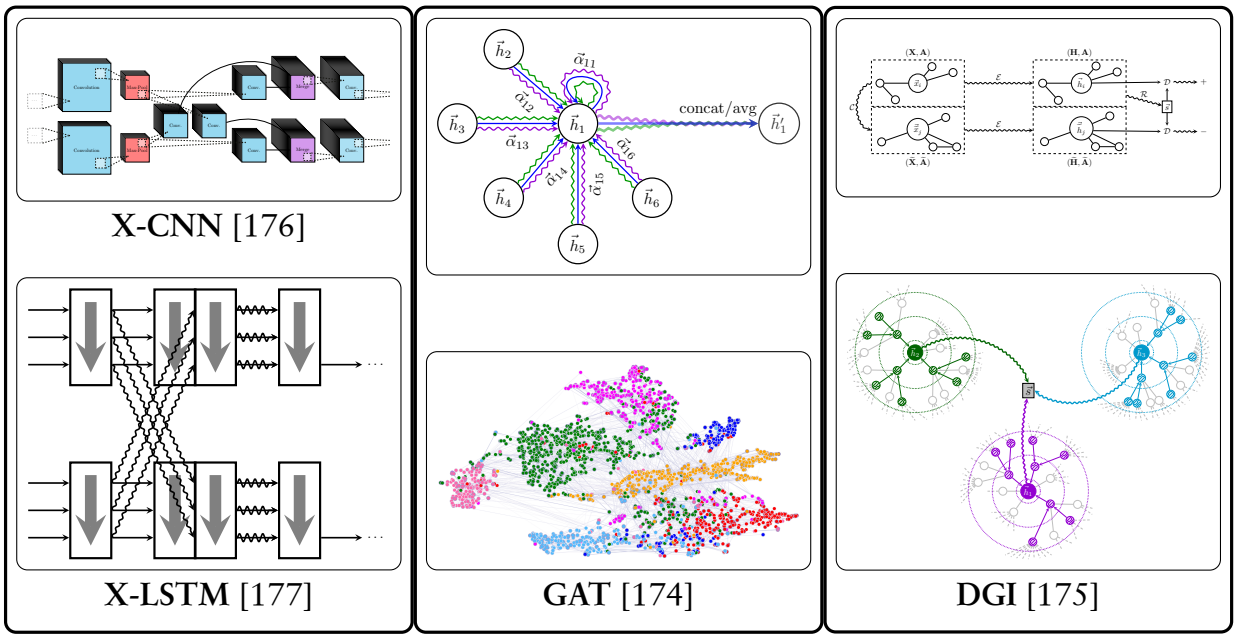
- Nontrivial structural inductive biases are introduced in the context of *multimodal deep learning* [124, 163]—wherein the assumption that different parts of the input should be processed by *different feature extractors* is encoded. Most work in this space focusses on *"late fusion"*; separately extracting features from each modality, and then driving decision-making using the combination of these features [2, 85], with recent work allowing for generic cross-modal interaction to occur at an earlier stage through *feature-wise transformations* [130].

- A substantial recent research direction involves generalising CNNs to operate on more general input structures—which grids are a special case of—such as *graphs* [49], *manifolds* [121] and *point clouds* [183]. Here, convolutions are generalised to more generic operations, often preserving appropriate locality, invariance and parameter sharing properties.

- *Relational inductive biases* [7] explicitly encode that a neural network should be mindful of *relations* present between *objects* in the input, and has recently been explored in the context of *physics interactions* [6], *visual question answering* [151], *multi-agent interactions* [75], and *relational memory modules* [150]. It should be noted that all of the above approaches assume a *complete graph* of relations; the tangential problem of *relational inference*, which aims to explicitly recover these relations and use them for inference, has also seen attention recently [90].

- Substantial work exists in introducing structural inductive biases in *deep reinforcement learning*, where they typically substantially improve the *few-shot* performance of the learnt agents. These approaches span *reintroducing symbolic methods* in deep RL algorithms [47], explicitly handling *relations between "objects"* present in the RL state [194], directly expressing *nontrivial structure of the agent* [149, 181], and allowing for information flow across tasks for *continual learning* [145, 146, 155]. In addition, structural biases have been successfully introduced in the context of *imitation learning* [37] and *programmable agents* [34].

As a few additional assorted topics where structural inductive biases have been deployed, I



Chapter 3
Early cross-modal fusion

Chapter 4
Graph convolutions

Chapter 5
Graph unsupervised learning

Figure 1.3: An overview of the main contributions presented in this dissertation. Firstly, two models with specialised structural inductive biases are proposed for *early fusion* in multimodal learning; one for grid-structured input modalities (**X-CNN** [176]) and one for sequential input modalities (**X-LSTM** [177]). Following, the desirable structural inductive biases for *graph convolutional layers* are outlined—and for the first time, simultaneously satisfied—in the graph attention network (**GAT** [174]) model. Lastly, local mutual information maximisation is successfully introduced—through the Deep Graph Infomax (**DGI** [175]) algorithm—as an *unsupervised learning objective* for graph-structured inputs, allowing a very powerful structural inductive bias to be introduced to learning node representations in conjunction with graph convolutional encoders.

would highlight *neural programmer-interpreters* [135], *amortised probabilistic programming inference* [138], *visual de-animation* [188], *capsule neural networks* [72, 147] and *compositional attention networks* [77].

1.3 Research questions and contributions

Having provided the necessary background overview of the importance of structural inductive biases, and surveyed the wide landscape of ongoing related work, I will now formulate *three* research questions that I seek to answer within this dissertation, along with the specific contributions made towards achieving them (and where they may be found in this document). Figure 1.3 may also be consulted for a condensed overview.

Q1. *Investigate viable candidate layers for **early fusion** in multimodal neural networks, and assess their practical deployability and benefits with respect to difficult learning environments, especially when the input data is sparse or incomplete.*

In Chapter 3 and [176, 177], I propose two cross-modal neural network architectures that perform early fusion between their modalities, operating on grid-like (**X-CNN**) and sequential (**X-LSTM**) input modalities, respectively. The methods rely on allowing separate modality streams to *exchange intermediate features*, therefore more easily exploiting any correlations between the modalities, and preserving the "unrestricted dataflow" property of fully-connected neural networks with a substantially smaller parameter count. It was shown that these methods can outperform their more traditional counterparts, especially under low training set sizes and incomplete input scenarios.

I will also highlight two related works that I have been involved in in an advisory role. One of them generalises the feature exchange to the 1D-2D case, setting strong results on audiovisual classification [21], while the other demonstrates that, while models such as the X-CNN provide an increase in hyperparameter count, these hyperparameters can be effectively tuned using an automated procedure [87].

Q2. *Study the generalisation of the **convolutional** operator from images to inputs exhibiting **graph structure** (i.e. the **graph convolutional layer**). Clearly outline the desirable properties for such an operator. Is it possible to satisfy all of these properties simultaneously, and do the theoretical properties imply competitive performance in practice?*

In Chapter 4 and [174], looking back at the favourable aspects of CNNs, I carefully specify the desirable properties for a *graph convolutional layer*, and assess why previously proposed such models needed to sacrifice some of these properties. I then define the *graph attention network* (**GAT**)[4], which generalises the *self-attention* operator [173] to the graph domain. Concluding that self-attention has *all* the desirable properties in this setting, I then deploy this model on several standard node classification benchmarks, and demonstrate competitive performance against other approaches (setting three state-of-the-art results at the time of submission).

Here I will also outline two subsequently released pieces of related work I've been involved in—both demonstrating that the proposed model holds up well in seemingly unrelated biomedical applications: cortical mesh-based parcellation [29] and

[4] **N.B.** The acronym GAT was chosen to avoid a name clash with generative adversarial networks [54].

neural paratope prediction [32].

Q3. *To what extent are graph convolutional networks useful for* ***unsupervised learning*** *on graph-structured data? Is it possible to usefully exploit* ***global*** *structural properties of a graph when formulating an unsupervised objective on it?*

In Chapter 5 and [175], I survey the previous approaches to unsupervised representation learning on graphs (primarily based on *random walks*), and realise that they are unlikely to be appropriate when used in combination with graph convolutional encoders. Building up on promising prior work on *local mutual information maximisation* in the image domain [73], I then propose the *Deep Graph Infomax* (**DGI**) learning algorithm for graph-structured inputs. This unsupervised objective allows every *local* component of the graph to be seamlessly mindful of the graph's *global* structural properties. As a result, the model is capable of producing node embeddings that are competitive with similar encoders trained using a *supervised* objective—even *exceeding* their performance at times!

Aside from exposing my primary research contributions—in the three chapters outlined above—this dissertation also contains (in Chapter 2) a comprehensive overview of background information on machine learning with deep neural networks. Particular emphasis is given on providing the essential mathematical details of the relevant models with structural inductive biases (going from CNNs and RNNs towards graph convolutional networks). I will also provide concluding thoughts, with particular emphasis on future work directions, in Chapter 6.

1.4 List of publications

Here I will provide the list of publications that have been used to convey the research contributions I have made and documented in this dissertation, along with making explicit the contributions of coauthors.

[176] **Veličković, P.**, Wang, D., Lane, ND. and Liò, P. (2016) X-CNN: Cross-modal convolutional neural networks for sparse datasets. *The 7th IEEE Symposium Series on Computational Intelligence (IEEE SSCI 2016)*.
Also presented at the *ARM Research Summit 2016*, the *ARM-Cambridge Research Showcase* and the *Fourth Edinburgh Deep Learning Workshop*.
`http://ieeexplore.ieee.org/document/7849978/`

[177] **Veličković, P.**, Karazija, L., Lane, ND., Bhattacharya, S., Liberis, E., Liò, P., Chieh, A., Bellahsen, O. and Vegreville, M. (2018) Cross-modal Recurrent Models for

Weight Objective Prediction from Multimodal Time-series Data. *The 12th EAI International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth 2018)*.
Also presented at the *2017 NIPS Workshop on Machine Learning for Health*, the *2017 NIPS Time Series Workshop* and *ObesityWeek 2017*.
https://dl.acm.org/citation.cfm?id=3240937

[21] Cangea, C., **Veličković, P.** and Liò, P. (2017) XFlow: 1D-2D Cross-modal Deep Neural Networks for Audiovisual Classification. *Workshop on Computational Models for Crossmodal Learning (CMCML) at The 7th Joint IEEE International Conference on Development and Learning and on Epigenetic Robotics (IEEE ICDL-EPIROB 2017)*.
Also presented at the *ARM Research Summit 2017*.
https://arxiv.org/abs/1709.00572

[87] Karazija, L., **Veličković, P.** and Liò, P. (2018) Automatic Inference of Cross-Modal Connection Topologies for X-CNNs. *The 15th International Symposium on Neural Networks (ISNN 2018)*.
https://link.springer.com/chapter/10.1007/978-3-319-92537-0_7

[174] **Veličković, P.**, Cucurull, G., Casanova, A., Romero, A., Liò, P. and Bengio, Y. (2018) Graph Attention Networks. *The 6th International Conference on Learning Representations (ICLR 2018)*.
https://openreview.net/forum?id=rJXMpikCZ

[29] Cucurull, G., Wagstyl, K., Casanova, A., **Veličković, P.**, Jakobsen, E., Drozdzal, M., Romero, A., Evans, A. and Bengio, Y. (2018) Convolutional neural networks for mesh-based parcellation of the cerebral cortex. *The 1st Conference on Medical Imaging with Deep Learning (MIDL 2018)*.
Also presented at the *2017 NIPS BigNeuro Workshop*.
https://openreview.net/forum?id=rkKvBAiiz

[32] Deac, A., **Veličković, P.** and Sormanni, P. (2018) Attentive cross-modal paratope prediction. *Journal of Computational Biology*.
Also presented at the *2018 ICML/IJCAI Workshop on Computational Biology*.
https://www.liebertpub.com/doi/10.1089/cmb.2018.0175

[175] **Veličković, P.**, Fedus, W., Hamilton, WL., Liò, P., Bengio, Y. and Hjelm, RD. (2019) Deep Graph Infomax. *The 7th International Conference on Learning Representations (ICLR 2019)*.
Also presented at the *2018 NeurIPS Relational Representation Learning Workshop*.
https://openreview.net/forum?id=rklz9iAcKQ

To summarise the contributions of individual authors:

- The majority of ideas, text, figures and experiments originated from the first author.
- The papers by Andreea Deac, Laurynas Karazija and Cătălina Cangea are extended versions of their Part II/III/MPhil projects (respectively) at the Computer Laboratory of the University of Cambridge, completed under my (co-)supervision.
- I have performed an advisory role on the paper by Guillem Cucurull, primarily advising on the deployment of graph convolutional networks to the studied task.
- Duo Wang, Guillem Cucurull, Arantxa Casanova and William Fedus contributed by running and proposing additional experiments over the papers they coauthored.
- Edgar Liberis and William L. Hamilton have made substantial contributions to the textual presentation and related work surveys over the papers they coauthored.
- As relevant domain experts, Angela Chieh, Otmane Bellahsen, Matthieu Vegreville, Konrad Wagstyl, Estrid Jakobsen, Alan Evans and Pietro Sormanni provided helpful domain-related input—and in some cases, the datasets used for the analysis—over all papers they coauthored.
- Pietro Liò, Nicholas D. Lane, Adriana Romero, R Devon Hjelm and Yoshua Bengio had an important advisory role over all papers they coauthored.

In addition, whenever this was possible, I have strived to make the source code of the relevant machine learning models and algorithms publicly available on my GitHub profile: `https://github.com/PetarV-`.

Chapter 2

Deep neural networks

> "Science is what we *understand well enough* to *explain* to a computer. Art is *everything else* we do."
>
> Donald E. Knuth

The contributions of this dissertation, outlined in Section 1.3, are exclusively centered around *deep neural network* architectures and optimisation methods. In order to provide necessary contextual information for interpreting their significance, in this chapter I will establish a common notational framework and comprehensively survey the most important background methods.

2.1 Mathematical notation

To aid clarity throughout the document, I will assume a consistent notation, to be described in this section, and referred to hereinafter. The layout idea is largely inspired by the notation used by [53].

s	Scalar
$\vec{v}$	Vector
$\mathbf{M}$	Matrix
$\mathbf{I}_n$	Identity matrix of shape $n \times n$
$\mathbf{I}$	Identity matrix of implied dimensionality
$\mathbb{R}$	The set of real numbers
$\{0, 1, 2\}$	The set containing 0, 1 and 2

(l, r)	The open interval from l to r
$[l, r]$	The closed interval from l to r
$f : \mathbb{X} \to \mathbb{Y}$	Function f, taking elements of set $\mathbb{X}$ to elements of set $\mathbb{Y}$
$f(x)$	f applied to input(s) x[1]
$\log x$	The natural logarithm of x
$\|\vec{v}\|$	The L_2 norm of $\vec{v}$
v_i	Element i of vector $\vec{v}$
$\vec{m}_i$	The i-th row vector of matrix $\mathbf{M}$
M_{ij}	Element (i, j) of matrix $\mathbf{M}$
$\mathbf{M}^T$	Transposed matrix $\mathbf{M}$
$\vec{a} \odot \vec{b}$	Elementwise product of $\vec{a}$ and $\vec{b}$
$\vec{a} \| \vec{b}$	Concatenation of $\vec{a}$ and $\vec{b}$
$\nabla_\theta y\|_{\theta=\theta'}$	Gradient of y with respect to θ, evaluated at $\theta = \theta'$
$\mathbb{P}(x)$	Probability distribution of a discrete RV x
$\mathbb{P}(x\|y)$	Probability distribution of a discrete RV x, given y
$p(x)$	Probability distribution of a continuous RV x
$x \sim \mathbb{P}(x)$	Random variable x sampled from $\mathbb{P}$
$\mathbb{E}_{x\sim\mathbb{P}(x)}[f(x)]$	The expected value of $f(x)$ given that x is sampled from $\mathbb{P}(x)$
$\text{Var}(f(x))$	The variance of $f(x)$
$\mathcal{U}(a, b)$	The uniform real distribution on the range $[a, b]$
$\mathcal{N}(\mu, \sigma)$	The normal distribution with mean μ and standard deviation σ

2.2 Essentials

In this section, I will recap the foundational material on deep neural networks by expressing their most potent variant, the *multilayer perceptron*, and covering all the relevant

[1] If f takes a scalar, and x is a tensor, then f is applied to all elements of x elementwise.

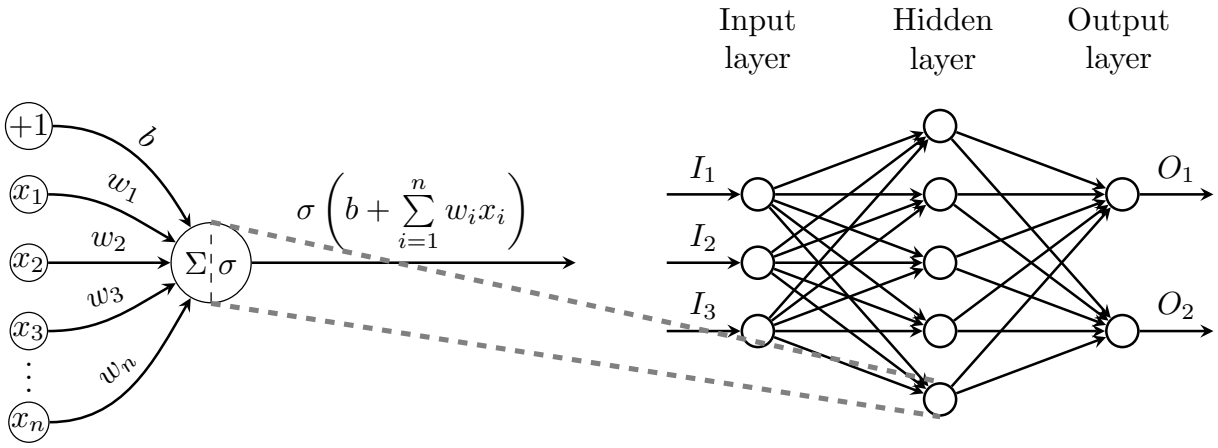


Figure 2.1: **Left:** A single perceptron model, with input vector $\vec{x}$, weights $\vec{w}$, bias b and activation function σ. **Right:** The dataflow of a small multilayer perceptron with a single hidden layer, highlighting the role of a single perceptron within it.

techniques for their training and regularisation (often applicable in all other settings).

2.2.1 The perceptron

Neural networks are machine learning models composed of simple interconnected processing units—the *(artificial) neurons* or *perceptrons* [140].

A perceptron (illustrated by Figure 2.1 (Left)) receives a vector $\vec{x} \in \mathbb{R}^n$ as *input*[2], and computes a *linear combination* of its entries. This combination is specified by a *weight vector*, $\vec{w} \in \mathbb{R}^n$, and a *bias value*, b. Afterwards, a nonlinear *activation function*, σ, is potentially applied to the result to obtain the final output value, $y \in \mathbb{R}$:

$$y = \sigma\left(b + \sum_{i=1}^{n} w_i x_i\right) = \sigma\left(b + \vec{w}^T \vec{x}\right) \tag{2.1}$$

The activation functions of interest for this dissertation (Figure 2.2) are as follows:

- The *logistic sigmoid* function:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{2.2}$$

  which monotonically maps real numbers to the $[0, 1]$ range, making it suitable for modelling Bernoulli random variables (e.g. for *binary classification*);

[2] The entries of $\vec{x}$ may be direct input values, or may be intermediate outputs of other perceptrons.

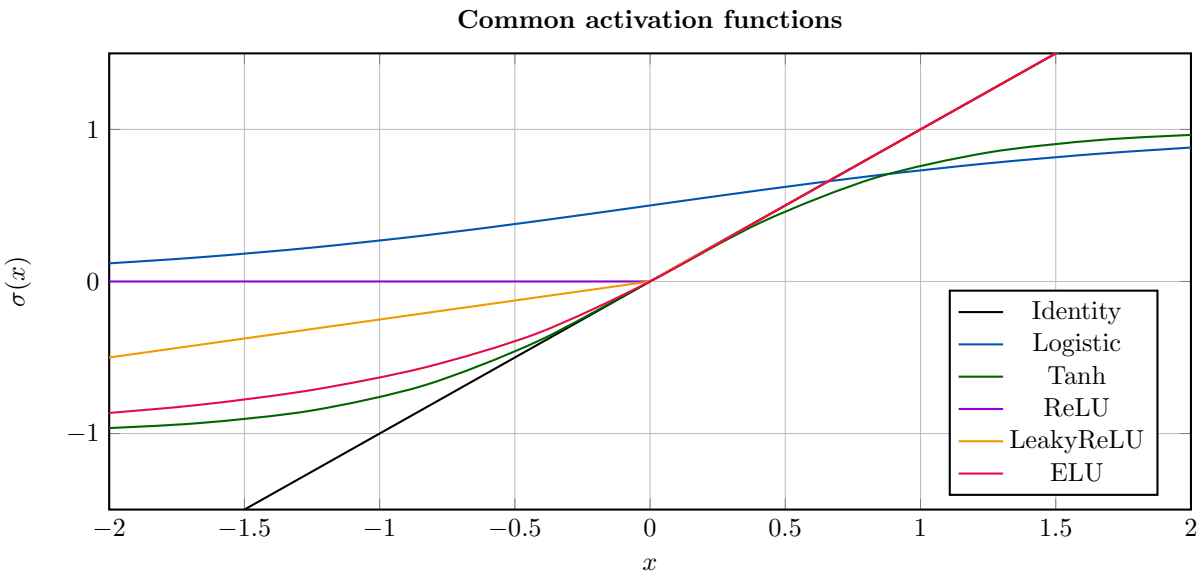


Figure 2.2: Plots of all the activation functions used throughout this dissertation.

- The *hyperbolic tangent* function:

$$\sigma(x) = \tanh(x) \tag{2.3}$$

  which can model general real outputs (given its $[-1, 1]$ range and sigmoidal shape).

- The *rectified linear* function (ReLU) [51]:

$$\sigma(x) = \max(x, 0) \tag{2.4}$$

  which is currently one of the most popular activations (owing to its biological plausibility, sparsity, lack of vanishing gradients, and computational efficiency).

- Two ReLU variants that extend its output signal into negative inputs:
  - The *leaky ReLU* [112], with a given negative slope, α:

$$\sigma(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases} \tag{2.5}$$

    called the *parametric ReLU* (PReLU) [67] when α is learnable (i.e. not fixed).
  - The *exponential linear unit* (ELU) [26], which thresholds the negative outputs:

$$\sigma(x) = \begin{cases} x & x \geq 0 \\ \exp(x) - 1 & x < 0 \end{cases} \tag{2.6}$$

2.2.2 Multilayer perceptrons

Neural networks typically assume a *feedforward* structure, organising their constituent neurons in a stack of *layers*, with each layer receiving the input(s) of previous layer(s) only. The most potent such architecture, a *multilayer perceptron* (**MLP**) (Figure 2.1 (Right)), *fully connects* the outputs of a layer to inputs of its succeeding layer. Namely, if we denote the input to an MLP layer as $\vec{x}$, then the j-th neuron of this layer computes the following:

$$y_j = \sigma\left(b_j + \vec{w}_j^T \vec{x}\right) \tag{2.7}$$

where $\vec{w}_j$ and b_j are the weights and bias of this neuron, respectively. For purposes of convenience, especially when leveraging deep learning frameworks, the computation of Equation 2.7 is often expressed in *matrix form*:

$$\vec{y} = \sigma\left(\mathbf{W}\vec{x} + \vec{b}\right) \tag{2.8}$$

where $\mathbf{W}$ and $\vec{b}$ are the *weight matrix* and *bias vector* of the MLP layer, respectively. The *depth* of an MLP may now easily be increased by stacking multiple such layers—each with their own parameters—e.g. like so:

$$\vec{y} = \sigma_2\left(\mathbf{W}_2\sigma_1\left(\mathbf{W}_1\vec{x} + \vec{b}_1\right) + \vec{b}_2\right) \tag{2.9}$$

If there's more than one *hidden* (intermediate) layer, the network is called a **deep** neural network. This is in relation to the *universal approximation theorem*, due to Cybenko [30]. It states that a neural network with one hidden layer and sigmoidal (e.g. logistic or hyperbolic tangent) activations can approximate *any* bounded continuous real function, arbitrarily precisely. Therefore, any network that is deeper provides no further theoretical power—however, it substantially simplifies *hierarchical* input processing (gradually extracting more complex features from the input). Furthermore, the proof of Cybenko's theorem is not constructive—it doesn't specify how many neurons this single hidden layer must have or how to obtain their weights and biases—and it is implied that in most cases the number of neurons is *prohibitively large*.

2.2.3 Learning the MLP parameters

Assume, for now, that we are working with an MLP structure that is *fixed*—i.e. that the number of neurons and activation functions for each of the layers are known. This implies that we know the dimensionality of $\mathbf{W}_i$ and $\vec{b}_i$, as well as the form of σ_i.

Specifying the values of $\mathbf{W}_i$ and $\vec{b}_i$ then fully determines the computation of the network.

In the following subsections, I will outline the most important steps for doing so.

2.2.3.1 Initialisation

Neural networks are typically optimised in a sequence of smaller steps that gradually adjust their parameters, starting from an *initial* value. Properly **initialising** a neural network is often critical, and can make the difference between good performance and *not converging at all*. The early successes of deep learning usually relied on *unsupervised pre-training* [42], either by leveraging *deep belief networks* (DBNs) [71] or *stacked autoencoders* [12, 134]. Currently, however, the gold-standard involves drawing each weight, i.i.d., from a carefully chosen probability distribution. Commonly, these are either the *uniform* or *normal* distribution with zero mean; that is, for each weight w_i, $w_i \sim \mathcal{U}(-x, x)$ or $w_i \sim \mathcal{N}(0, \sigma)$. This means that only the *variance* of the weights, σ^2, needs to be specified in order to determine the initialisation scheme.

Assuming a *linear* neuron with zero-mean and uncorrelated inputs[3] $\vec{x}$ and weights $\vec{w}$, consider the variance of the output of the neuron:

$$\text{Var}\left(\sum_{i=1}^{n_{\text{in}}} w_i x_i\right) = \sum_{i=1}^{n_{\text{in}}} \text{Var}(w_i x_i) = \sum_{i=1}^{n_{\text{in}}} \text{Var}(W)\text{Var}(X) = n_{\text{in}}\sigma^2\text{Var}(X) \tag{2.10}$$

where n_{in} is the *fan-in*, the number of inputs to the neuron. In order to have consistent gradient updates, it is useful if this neuron (at least early-on during training) *preserves the variance of the input*. Therefore, we should set $\sigma^2 = \frac{1}{n_{\text{in}}}$, obtaining *LeCun initialisation* [104]. A similar argument for preserving the variance of the computed weight updates (which are computed *backwards* from the output layers towards the input), yields the condition $\sigma^2 = \frac{1}{n_{\text{out}}}$, where n_{out} is the *fan-out*, the number of neurons directly receiving this neuron's output. As it is typically infeasible to satisfy both conditions at once, we may elect to satisfy their *average*:

$$\sigma^2 = \frac{2}{n_{\text{in}} + n_{\text{out}}} \tag{2.11}$$

leading to *Xavier initialisation* [50], which is now the standard initialisation for linear and sigmoidal neurons (as sigmoidal functions are roughly linear in their *unsaturated* range). A similar argument for the ReLU-like functions leads to the following condition:

$$\sigma^2 = \frac{2}{n_{\text{in}}} \tag{2.12}$$

[3] **N.B.** This assumes that we have *standardised* our inputs upfront, i.e. $\mathbb{E}(X) = 0$ for all input values x_i.

known as *He initialisation* [67], which was critical to surpassing human performance on the ImageNet benchmark.

2.2.3.2 Optimisation

Once weights have been appropriately initialised, a learning algorithm is applied to iteratively fine-tune them to the specifics of the task at hand. This is usually done by expressing a differentiable *loss function*, $\mathcal{L}$, which dictates the suitability of the network's parameters for the task, and is the primary objective for a (typically first-order) optimisation algorithm. This objective is usually *data-driven*.

As a guiding example, I will use the standard *supervised learning* setup: assume we have a *training set*, $\vec{s} = \{(\vec{x}_i, \vec{y}_i)\}_{i=1}^{m}$ of m input-output pairs. Feeding the input $\vec{x}_i$ into the network, we obtain the prediction of the network on this input, $\vec{\hat{y}}_i$, as the output of its final layer. For simple *regression* problems, we may use the *mean squared error* of this prediction against the ground truth $\vec{y}_i$, over the entire training set, as the loss function:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{m} \sum_{i=1}^{m} \left\| \vec{y}_i - \vec{\hat{y}}_i \right\|^2 \tag{2.13}$$

If, instead, we are dealing with *classification* problems, then the ground-truth $\vec{y}_i$ is usually a one-hot probability distribution over the K classes. To interpret the network predictions, $\vec{z}_i$, as probability distributions, they are passed through the *softmax* function:

$$\mathbb{P}\left(\vec{x}_i \in \text{class } j\right) = \hat{y}_{ij} = \frac{\exp(z_{ij})}{\sum_{k=1}^{K} \exp(z_{ik})} \tag{2.14}$$

After this, the distance between the two distributions is commonly expressed using the *cross-entropy* loss function:

$$\mathcal{L}_{\text{CE}} = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{K} y_{ij} \log \hat{y}_{ij} \tag{2.15}$$

For deep neural networks, the loss function, $\mathcal{L}$, is commonly optimised using *mini-batch stochastic gradient descent* (SGD). At each iteration, a mini-batch, $\mathcal{B}$ of m training examples is selected, and the *backpropagation* algorithm [143] is applied to them in order to compute the *gradient* of the loss function, $\nabla_{\vec{\theta}} \mathcal{L}_{\mathcal{B}}$ with respect to the network's parameters, $\vec{\theta}$. This computation involves repeatedly applying the *chain rule* for partial derivatives, proceeding progressively *backwards* from the output layer towards the input layer of the

network. The parameters are then updated by doing a single step of gradient descent:

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t - \eta \nabla_{\vec{\theta}} \mathcal{L}_{\mathcal{B}}\big|_{\vec{\theta}=\vec{\theta}_t} \tag{2.16}$$

where η is the *learning rate*. The choice of learning rate is critical to convergence rate and stability. However, recent advances in *adaptive optimisers*—that dynamically adjust the learning rate based on historical gradient updates—have substantially reduced this issue. At the time of writing this document, the *Adam* (adaptive moment) SGD optimiser [88] is the most popular variant. Letting $\vec{g}_t = \nabla_{\vec{\theta}} \mathcal{L}_{\mathcal{B}}\big|_{\vec{\theta}=\vec{\theta}_t}$ be the gradient at time t, the algorithm maintains exponential moving averages of both $\vec{g}_t$ and $\vec{g}_t^2 = \vec{g} \odot \vec{g}$:

$$\vec{m}_{t+1} = \beta_1 \vec{m}_t + (1 - \beta_1)\vec{g}_t \qquad \vec{v}_{t+1} = \beta_2 \vec{v}_t + (1 - \beta_2)\vec{g}_t^2 \tag{2.17}$$

The default values of the mixing constants are $\beta_1 = 0.9$ and $\beta_2 = 0.999$. This heavily biases the averages towards zero early on, so the following *bias correction* is applied:

$$\hat{\vec{m}}_t = \frac{\vec{m}_t}{1 - \beta_1^t} \qquad \hat{\vec{v}}_t = \frac{\vec{v}_t}{1 - \beta_2^t} \tag{2.18}$$

Finally, these averages are used to update the network parameters, effectively giving each parameter, θ_i, a "bespoke" update corresponding to its average ($\hat{m}_i$), scaled depending on its squared average ($\hat{v}_i$):

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t - \eta \frac{\hat{\vec{m}}_t}{\sqrt{\hat{\vec{v}}_t} + \epsilon} \tag{2.19}$$

where η is the *initial learning rate* (which may now be set far more leniently), and ϵ is a small constant (usually 10^{-8}) designed to protect against division by zero.

In practice, mini-batch SGD is often performed by first shuffling the entire training set, then sequentially taking batches from the shuffled dataset. Once the entire dataset is covered in this way, a single *epoch* of training is completed, and the dataset is reshuffled in advance of the next one.

2.2.3.3 Regularisation

Deep neural networks are usually *overparametrised* models, as they are often optimised on training datasets of substantially smaller sizes than their parameter counts—that are often in the *millions*. This makes them extremely prone to *overfitting*; the effect of learning to *memorise the training set*, along with any noise present within it. As a consequence, overfitted models fail to generalise outside of the training set (a defining requirement for well-trained machine learning models)—see Figure 2.3.

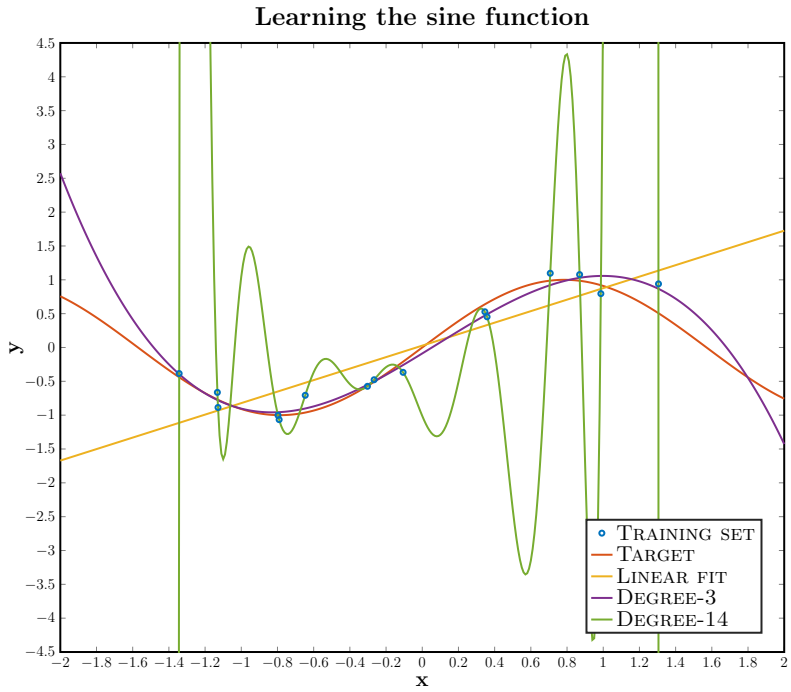


Figure 2.3: Potential pathological behaviours of learning systems, illustrated on the simple example of fitting a sine wave from a (noisy) training set of points. A pure linear fit is not powerful enough to model the data's curvature (and therefore **underfits**). A degree-14 polynomial fits the training data perfectly, but captures all of its inherent noise and catastrophically fails to generalise (and therefore **overfits**).

To protect neural networks against overfitting, **regularisation** is usually the first line of defence. Broadly speaking, it requires the neural network to optimise for the same problem, under *additional constraints* that somehow restrict the set of parameters available to it during training—hopefully, in a way that discourages memorisation. For the purposes of the models deployed within this dissertation, *three* regularisation techniques have been utilised, and I will describe them in turn, across separate paragraphs.

L_2-regularisation This regulariser imposes a prior belief that the weights and biases of the network should not *deviate too far from zero*. Aside from constraining the model complexity, this may also yield clear numerical benefits. It is implemented as a *weight penalty* term—based on the L_2 norm—which is added to the loss function as follows:

$$\tilde{\mathcal{L}} = \mathcal{L} + \frac{\lambda}{2}\|\vec{\theta}\|^2 \tag{2.20}$$

where λ is a constant that controls the importance of penalising the weights compared to optimising the loss function, and should be chosen carefully.

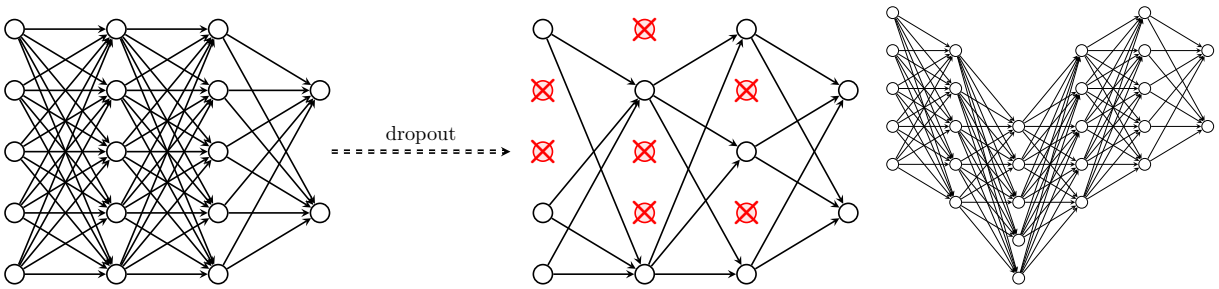


Figure 2.4: **Left:** The effect of a single iteration of dropout on a fully-connected neural network. **Right:** An illustration of the internal covariate shift effect—as training progresses, the statistics (mean/standard deviation) of the output of an intermediate neural network layer begin to deviate from other layers.

Dropout When trained without additional constraints, neural networks will tend to encourage *highly specialised neurons*. Even worse—there may be neurons that perform poorly, with several other neurons present solely to *correct for its mistakes*. As a consequence, the network becomes *overreliant* on individual neurons, and failure of a single neuron may compromise the operation the entire network.

The *dropout* [162] technique aims to mitigate this overreliance, by randomly "killing" each neuron in a layer with probability p, during training only (Figure 2.4 (Left)). To preserve the expected activation value, outputs of surviving neurons are scaled by $\frac{1}{1-p}$.

Batch normalisation The initialisation techniques covered in Section 2.2.3.1 have been concerned with *preserving the statistics of the input signal* as it propagates through the network. However, as training progresses, especially for deeper networks, it is to be expected that weights are pushed in a direction that will change the intermediate layer statistics—an effect known as the **internal covariate shift** (Figure 2.4 (Right)). This is an important problem, as neurons need not only to perform a particular feature-extracting function, but also to *adapt to input statistics that are changing over time*.

A practical solution turns out to be quite simple—simply periodically *renormalising* the activations of the network. The first—and still most widely used—approach to doing so is to renormalise across each training minibatch. This approach is known as *batch normalisation* [78]. If we let the outputs of a layer across a minibatch be $\mathcal{B} = \{\vec{x}_1, \ldots, \vec{x}_m\}$, we renormalise as follows, to obtain new outputs, $\vec{y}_i$:

$$\vec{\mu}_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}\vec{x}_i \qquad \vec{\sigma}^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}(\vec{x}_i - \vec{\mu}_{\mathcal{B}})^2 \tag{2.21}$$

$$\vec{\hat{x}}_i = \frac{\vec{x}_i - \vec{\mu}_{\mathcal{B}}}{\sqrt{\vec{\sigma}^2_{\mathcal{B}} + \epsilon}} \qquad \vec{y}_i = \gamma\vec{\hat{x}}_i + \beta \tag{2.22}$$

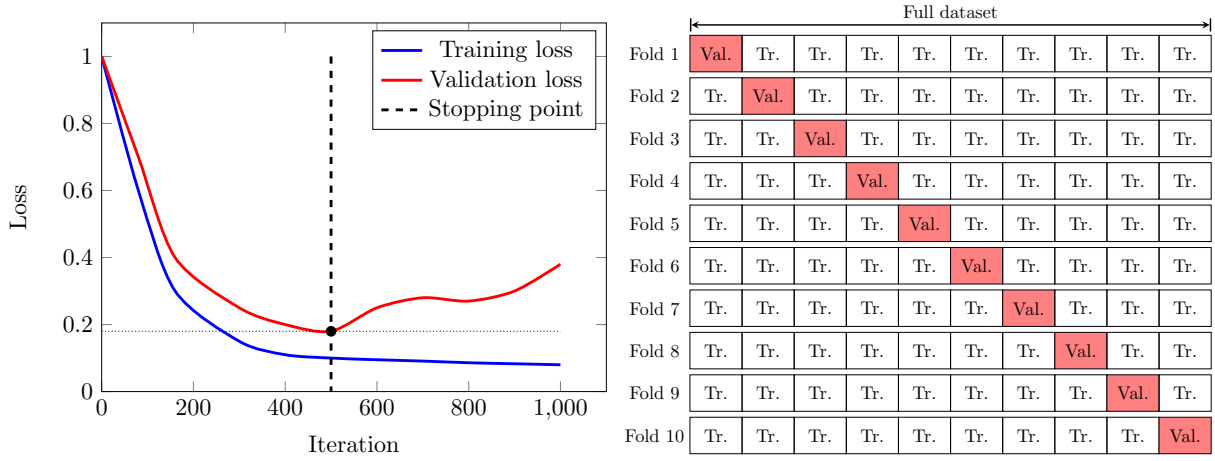


Figure 2.5: **Left:** An illustration of early stopping, with a patience of 500 iterations. **Right:** 10-fold crossvalidation. At each fold, a distinct partition of the dataset is used for validation, with the remainder used for training. Results are averaged across all folds.

Here, γ and β are *learnable* parameters. As the normalisation operation may restrict the predictive power of the neural network[4], this allows the network to *revert* to the old values, if it is beneficial to do so. Interpreted from an alternate angle, the network is allowed to *learn its own normalisation scheme*.

At test time, as we should not observe the remainder of the test batch, the values of $\vec{\mu}_{\mathcal{B}}$ and $\vec{\sigma}^2_{\mathcal{B}}$ are taken across the *entire training set*—in practice, this is usually computed as an exponential moving average during training.

2.2.4 Hyperparameter tuning

In the preceding section, we have assumed many aspects of the network architecture and setup to be known and fixed—parameters such as the network depth, number of neurons per layer, learning rate, dropout probability, etc. All of these parameters—not explicitly optimised by SGD—are hence referred to as *hyperparameters*. Appropriately choosing them—especially in an automatic way—is an open area of research.

For the purposes of my contributions, all hyperparameters are optimised using a held-out *validation set*. To choose a hyperparameter configuration out of a set of possible configurations, the network is trained from scratch with each one, and the configuration that achieves the best performance metric on the validation set is retained.

A special case of this is *early stopping* (Figure 2.5 (Left)), where the validation set is

[4] For example, if the values are fed into a sigmoidal activation function, normalisation may persistently restrict them to the *unsaturated* region of the input space.

used to optimise the *number of iterations* taken by SGD. Typically, if the loss (or any other appropriate metric) on the validation set has not improved in a certain number of iterations or epochs (known as the *patience*), training is prematurely terminated.

Lastly, if the training set is not too large, it is possible to utilise its entirety for the purposes of validation, by using the k*-fold crossvalidation* method. Here, the entire dataset is partitioned into k parts (or *folds*), and at each iteration, $k-1$ folds are used for training with the final one used for validation (Figure 2.5 (Right)). The overall crossvalidation performance is taken as the average of performance across the individual folds. Typically, the crossvalidation is performed in a *stratified* fashion, meaning the classes present in the individual folds' examples should respect the class distribution of the entire dataset.

2.3 Structured neural networks for grids, sequences and sets

Despite its potency, the multilayer perceptron (as described thus far) is primarily designed for processing *flat* (unstructured) data. To make further progress on larger-scale inputs, we need to exploit simple *structure* in the inputs whenever possible. Here I will present three popular directions for doing so—for when the data is *grid-like* (e.g. images), *sequential* (e.g. sound or text), or consists of *unordered sets* (e.g. point clouds).

It should be noted that each of the architectures to be presented can be re-expressed as a (constrained) MLP; however, an MLP would require substantially more training data to rediscover the structural inductive biases present therein.

2.3.1 Convolutional neural networks

When working with image(-like) data in neural networks, they will typically be represented as tensors[5] of shape $h \times w \times d$, with d image *channels* of height h and width w. As MLPs treat each input element *independently*, this means that $O(h \times w \times d)$ parameters are introduced per each neuron in the first hidden layer. As images become larger, the memory requirements and tendency for overfitting quickly increase—with datasets on the scale of $32 \times 32 \times 3$ (e.g. CIFAR-10 [97]) already becoming too challenging for such unrestricted architectures to handle.

The general idea for handling this (in the case of image classification, at least) is straightforward: *downsample* the image to a size which is small enough to be processed by an MLP. However, pure downsampling potentially discards a wealth of useful information,

[5] This can easily be generalised to more (or less) than two input dimensions.

Figure 2.6: The power of the *convolutional* operator, demonstrated on the *Lena* image used for computer vision benchmarking. With a very small amount of parameters (e.g. 3×3 kernel matrices), convolutions are capable of extracting valuable information such as *edges* out of an image (which often encode most of the valuable information therein).

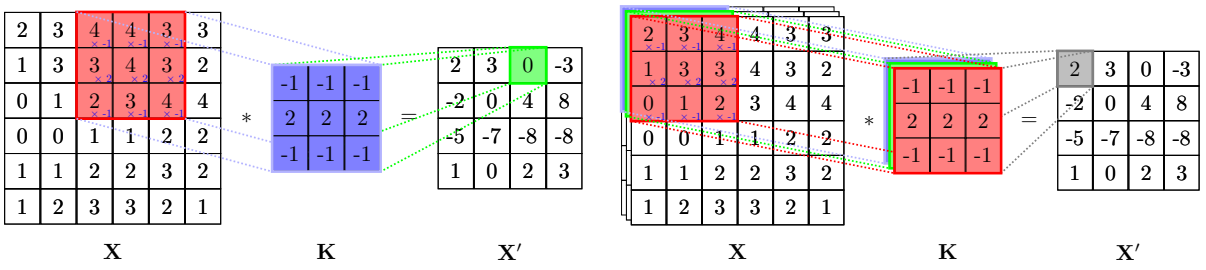


Figure 2.7: A convolutional layer, as applied on a *single-channel* image (Equation 2.23; **left**) and a *three-channel* (e.g. RGB) image (Equation 2.24; **right**).

and therefore it would be highly beneficial if we could extract some useful image features first, before downsampling it too strongly. In order to make a *parameter-efficient* feature extractor for images, we need to exploit the *spatial structure* present in such inputs—a natural candidate for such a layer is the *convolutional* operator (Figure 2.6).

2.3.1.1 Convolutional layers

To define a convolutional operator, we will consider the case of 2D inputs (i.e. single-channel images), $\mathbf{X} \in \mathbb{R}^{h \times w}$. Let $\mathbf{K} \in \mathbb{R}^{n \times m}$ be a small *kernel matrix* (typically, $n = m = 3$ in modern neural network architectures [160]). The kernel is overlaid in all possible ways over the input (see Figure 2.7 (Left)), recording sums of elementwise products to create a new image, $\mathbf{X}'$:

$$X'_{ab} = \sum_{i=1}^{n} \sum_{j=1}^{m} K_{ij} \cdot X_{a+i-1,b+j-1} \tag{2.23}$$

It should be noted that this approach is actually *cross-correlation* rather than the convolution; but, for purposes of machine learning, the two are equivalent. The image is typically *padded* with sufficiently many zeroes on the edges, to ensure that the size of the

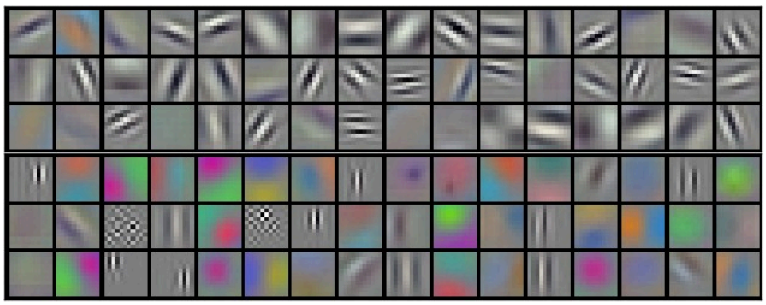


Figure 2.8: Visualisation of learnt convolutional kernels for the first layer of AlexNet [98], an early success story of deep neural networks for computer vision, after training on ImageNet. The majority of the kernels can be easily related to *edge detectors* (in various directions). Note that the model was never *instructed* to look for edges.

output matches the size of the input.

Moving from here to a convolutional *layer* requires a few extensions, primarily to support multiple channels:

- If the input has multiple channels, the kernel matrix is *extended* in the channel dimension to match the input—this *kernel tensor* specifies a single output channel;
- Each output channel requires a separate kernel tensor;
- Lastly, a channel-specific bias and nonlinearity may be applied to obtain the final output image (sometimes known as a *feature map*).

In summary, to process an input image $\mathbf{X} \in \mathbb{R}^{h \times w \times d}$ using a convolutional layer computing d' channels, we require a kernel tensor $\mathbf{K} \in \mathbb{R}^{n \times m \times d \times d'}$ and a bias vector $\vec{b} \in \mathbb{R}^{d'}$, and proceed as follows (Figure 2.7 (Right)):

$$X'_{abc} = \sigma\left(b_c + \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{d} K_{ijkc} \cdot X_{a+i-1,b+j-1,k}\right) \tag{2.24}$$

This layer clearly exploits the structure present in an image—with neighbouring pixels influencing each other far more strongly than ones on opposite corners. The operator is also *translation invariant*—as the same kernels are applied identically across each image patch, we are encoding a structural bias that a feature of interest is important, *no matter where it occurs in the image*.

As the sole computation being performed is multiplication by weights and summation, the identical *gradient descent* training from before may be performed to find suitable kernels. Often, these kernels will learn to do highly interpretable operations, with the

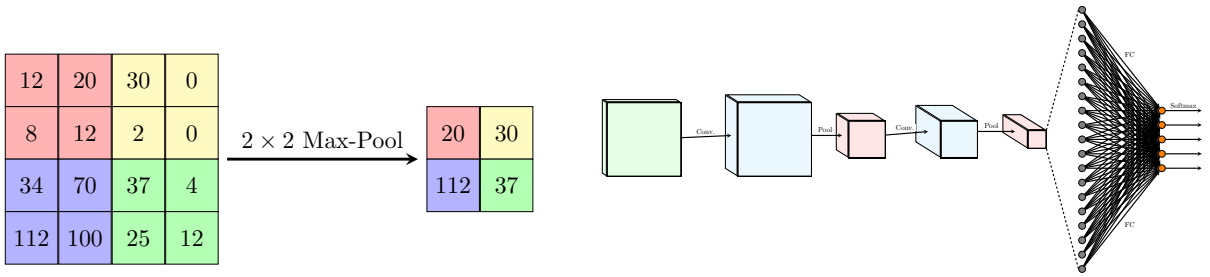


Figure 2.9: **Left:** Illustration of the 2×2 max-pool operator, applied on a single-channel image. **Right:** An overview of a typical CNN architecture for image classification. Convolutional and pooling layers are interleaved, with the depth (number of features computed per pixel) increased at the expense of height and width. Once the images are small enough, they are *flattened* into 1D, and an MLP is used to classify them.

first-layer ones almost always becoming various *edge detectors* (Figure 2.8).

Convolutional layers specify several new hyperparameters, with the *output channel count*, d', often the most important one to optimise, and others taking known default values.

2.3.1.2 Pooling layers

Once a sufficient number of convolutions has been applied to the input, we may proceed to downsample the image. As convolutions *"light up"* (return high values) when a certain pattern of interest is detected in the input, when summarising the image it makes sense to preserve *maximally activated* components of it. This motivates the *max-pooling* layer, which takes $n \times m$ (usually disjoint, with $n = m = 2$) image patches across each channel and summarises them by taking only the maximal pixel value (Figure 2.9 (Left)).

Note that a 2×2 max-pooling layer (which is most common) will preserve only a quarter of the pixels, while leaving the channel axis unchanged. This provides more space for gradually *increasing the channel size*, and therefore computing a richer quantity of higher-level features. A typical convolutional neural network architecture thus consists of a series of interleaved convolutional and pooling layers, until the input is reduced sufficiently to be processed by an MLP—as in Figure 2.9 (Right).

2.3.1.3 Residual networks

It is generally perceived that, for image recognition tasks, deep convolutional networks strongly draw their outperformance from both their depth **and** their parameter tying [172]. However, training very deep convolutional networks used to be notoriously difficult. As shown in [68], overly deep convolutional networks *do not overfit*, as their training performance degrades along with their testing performance. This is somewhat

counterintuitive, as increased depth not only comes with larger parameter spaces to optimise, but moreover, if the additional layers are useless to learning good representations, the layer could simply choose to *ignore* them (by learning the **identity** function).

Through this counterintuitive property, the authors proposed an obvious fix: the *residual (skip) connection* [68], which "shortcuts" a particular block of operations in a neural network. If a neural network block computes a function $\Phi(\vec{x})$, a skip connection allows $\vec{x}$ to directly reach the output of this block through aggregation; i.e. the neural network simply computes $\Phi(\vec{x}) + \vec{x}$. More generally, if the block changes the input dimensionality, a simple linear projection may be introduced to correct for this:

$$\widetilde{\Phi}(\vec{x}) = \Phi(\vec{x}) + \mathbf{W}_{\text{skip}}\vec{x} \tag{2.25}$$

Note that, effectively, this operation allows signals to directly reach deeper stages of the input processing, and therefore *allows the network to choose its own depth*. This turned out to be a very powerful and versatile concept, that enabled a 152-layer network to win the ImageNet competition in 2015. Currently, it is ubiquitous in deep learning.

2.3.2 Recurrent neural networks

Now, consider the situation in which the inputs are *sequential* (e.g. text or speech)—consisting of arbitrarily many *steps*, wherein at each step we have a fixed set of input features[6]. As fully-connected layers expect a *fixed-size* input, they will no longer suffice.

Convolutional layers (in 1D), treating the input step features as individual channels) may still be applicable. However, their parameter sharing properties mean that they are unable to efficiently *summarise* sequences of widely different sizes[7].

Hence, a neural network layer that is simultaneously capable of **handling** and rapidly **summarising** *variable-length sequential input*—through exploiting appropriate structural inductive biases—is highly desirable. Currently, *recurrent neural networks* represent the key development in this space, and therefore they will be the main focus of this subsection.

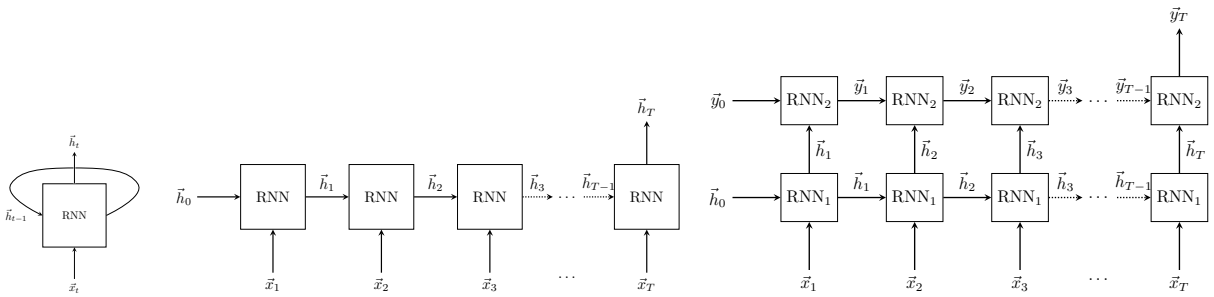


Figure 2.10: **Left:** A single recurrent neural network cell. **Middle:** Unrolling of the RNN cell in order to perform backpropagation through time (**N.B.** each "RNN" block has *same* parameters). **Right:** A "deep" RNN, obtained by stacking two RNN cells.

2.3.2.1 The recurrent layer

Consider a sequential input, $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_T\}$, consisting of T steps[8]. At each step, a **recurrent neural network** (RNN) will compute a *summary*, $\vec{h}_t$, of all input steps up to and including position t. This (partial) summary is computed conditional on the current step's features and the previous step's summary, through a shared function f (Figure 2.10 (Left)):

$$\vec{h}_t = f(\vec{x}_t, \vec{h}_{t-1}) \tag{2.26}$$

Specially, the initial summary vector is usually set to the zero-vector, i.e. $\vec{h}_0 = \vec{0}$. This form of *temporal weight sharing* encodes another *translation invariance* assumption; namely, that a pattern of interest is equally important, regardless of *when it happens in the sequence*. While this introduces *loops* in the network's computational graph, in practice the network is *unrolled* for an appropriate number of steps—see Figure 2.10 (Middle)—allowing for *backpropagation through time* to be applied.

The summary vectors may then be appropriately leveraged for the downstream task—if a prediction is required at every step of the sequence, then a shared MLP may be applied to each $\vec{h}_t$ individually. For classifying entire sequences, typically the final summary, $\vec{h}_T$, is passed through an MLP. For arbitrary *sequence-to-sequence* translation tasks [166] (such as machine translation), $\vec{h}_T$ may be used as an initial input for a *decoder RNN*, with outputs of RNN blocks being given as inputs for the next step.

Lastly, it should be noted that it is easy to *stack multiple RNNs*—simply use the $\vec{h}_t$ vec-

[6] For simplicity, we will assume that these features are *flat*. However, it is often straightforward to extend this framework to cases where each step's features are more complicated (e.g. in the case of *video* input, wherein each step consists of an input image).

[7] It should be noted that recent advances in *à trous* (dilated) convolutional layers have given convolutions substantially faster *input coverage*, making them competitive in a variety of sequential tasks traditionally dominated by recurrent neural networks [32, 86, 127].

[8] T may be different across different input sequences.

tors as an input sequence for a second RNN. This kind of construction, depicted in Figure 2.10 (Right), is occasionally called a "deep RNN", which is potentially misleading. Effectively, due to the repeated application of the recurrent operation, even a single RNN "layer" has depth *equal to the number of input steps*. This introduces a kind of learning problem uniquely challenging for RNNs—to be discussed in the following paragraph.

2.3.2.2 SimpleRNN and vanishing gradients

As can be seen from Equation 2.26, the choice of the function f (the **recurrent cell**) is very flexible—the sole requirement is for it to consume two vector inputs (current step and previous summary) and produce a single vector output (current summary).

In the simplest case, this role may be performed by a single-layer MLP; and this is exactly what is done by the original **SimpleRNN** model [41, 82]. Its computation is performed as follows:

$$\vec{h}_t = \sigma\left(\mathbf{W}\vec{x}_t + \mathbf{U}\vec{h}_{t-1} + \vec{b}\right) \tag{2.27}$$

where $\mathbf{W}$ and $\mathbf{U}$ are learnable weight matrices, $\vec{b}$ is a learnable bias, and σ is a nonlinearity.

While this model can be quite powerful for many tasks of interest [100], it suffers from potentially pathological learning dynamics. As just mentioned, even "single-layer" SimpleRNNs become *very deep* neural networks when trained on long-range sequences. When using backpropagation through time, a gradient with respect to a particular input step will depend on a *product* of gradients across all subsequent steps, multiplied by the network output at the current step.

Consider a SimpleRNN with a sigmoidal function for σ. The magnitude of the derivative of σ is then *always* between 0 and 1, and multiplying many such values results in gradients that quickly *tend to zero*, implying that early steps in the input sequence may not be able to have influence in updating the network parameters at all. This is the **vanishing gradient problem**, which is especially pathological for recurrent neural networks. For example, consider the next-word prediction task (common in e.g. predictive keyboards), and the input text *"Petar is Serbian. He was born on ... [long paragraph] ... Petar currently lives in _____"*. Here, predicting the next word as *"Serbia"* may only be reasonably concluded by considering the very start of the paragraph—but gradients have *vanished* by the time they reach this input step.

Deep MLPs have also suffered from the vanishing gradient problem until the invention of the ReLU activation (which has gradients equal to exactly zero or one—thus fixing

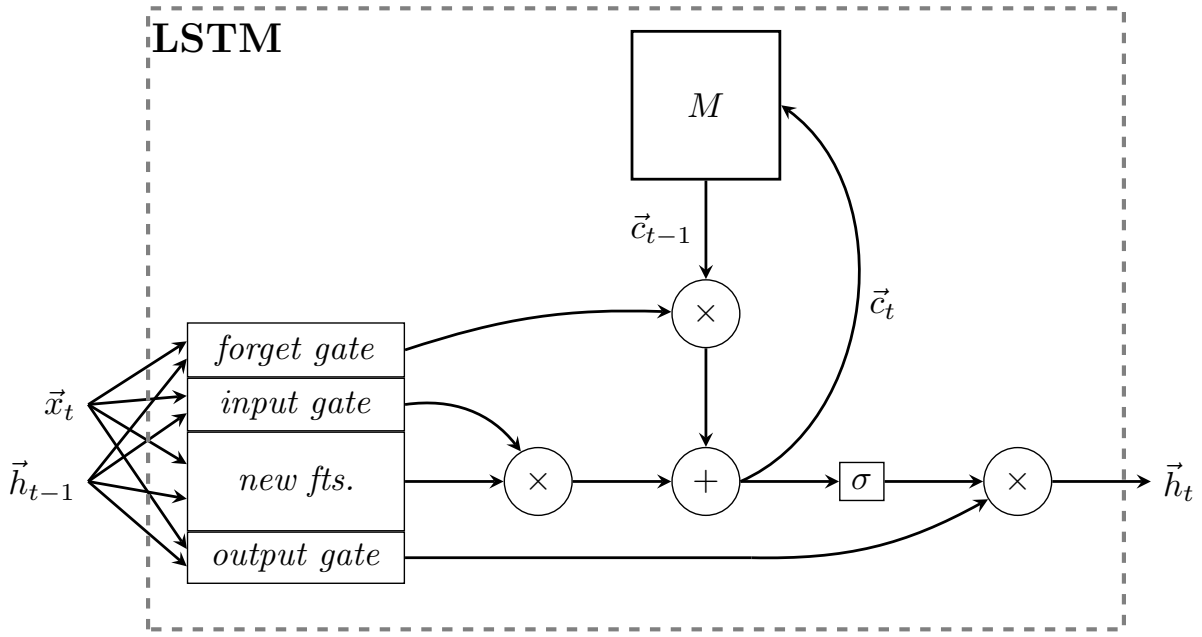


Figure 2.11: The dataflow of the long short-term memory cell, with its memory cell (M) and gating mechanisms clearly highlighted.

the vanishing gradient problem). However, in RNNs, using ReLUs may easily lead to *exploding gradients*, as the output space of the cell is now unbounded, and gradient descent will update the cell once for every input step, quickly building up the scale of the updates.

2.3.2.3 Long short-term memory

Most developments in recurrent cells since the SimpleRNN tackle the vanishing gradient problem, often demonstrating highly promising results. The first such model that achieved high popularity is the long short-term memory (LSTM) [74]. Despite several other layers being proposed in recent times (such as the *gated recurrent unit* [24]), none of them have been able to improve on the LSTM on average. As such, in this subsection I will solely consider the LSTM.

The LSTM augments the recurrent computation by introducing a *memory cell*, which stores vectors $\vec{c}_t$ that are *preserved* between computational steps—thus exposing a gradient of 1, eliminating the vanishing gradient problem. The LSTM cell's outputs, $\vec{h}_t$ are computed based on these vectors, and they are in turn computed using $\vec{x}_t$, $\vec{h}_{t-1}$ and $\vec{c}_{t-1}$. Critically, the cell is not *completely overwritten* based on $\vec{x}_t$ and $\vec{h}_{t-1}$, which would expose the network to the same issues as the SimpleRNN. Instead, a certain quantity of the previous cell state may be *retained*—and the proportion by which this occurs is *explicitly learned through data*.

Explicitly, the LSTM cell leverages *four* single-layer MLPs on $\vec{x}_t$ and $\vec{h}_t$ (Figure 2.11)

to perform the following computation ($\mathbf{W}_*$, $\mathbf{U}_*$ and $\vec{b}_*$ are appropriate learnable weights and biases):

- Analogously to SimpleRNN, the *"new features"* MLP computes the *candidate feature vector* to be written into the memory cell:

$$\vec{\tilde{c}}_t = \tanh\left(\mathbf{W}_c\vec{x}_t + \mathbf{U}_c\vec{h}_{t-1} + \vec{b}_c\right) \tag{2.28}$$

  Note that, as the LSTM is optimised to combat vanishing gradients, a sigmoidal function (such as the hyperbolic tangent) is now appropriate.

- The *input gate* computes the proportion of the candidate vector to be actually allowed into the memory cell:

$$\vec{i}_t = \text{logistic}\left(\mathbf{W}_i\vec{x}_t + \mathbf{U}_i\vec{h}_{t-1} + \vec{b}_i\right) \tag{2.29}$$

  where the logistic sigmoid nonlinearity gives values in the range $[0, 1]$, thus controlling the proportion of candidate features.

- The *forget gate* computes the proportion of the previous cell state to be retained:

$$\vec{f}_t = \text{logistic}\left(\mathbf{W}_f\vec{x}_t + \mathbf{U}_f\vec{h}_{t-1} + \vec{b}_f\right) \tag{2.30}$$

- The *output gate* computes the proportion of the final output vector to be allowed to exit the recurrent cell:

$$\vec{o}_t = \text{logistic}\left(\mathbf{W}_o\vec{x}_t + \mathbf{U}_o\vec{h}_{t-1} + \vec{b}_o\right) \tag{2.31}$$

Once all these values are computed, the memory can first be updated, respecting the proportions dictated by input and forget gates. Then the output of the recurrent cell is computed, respecting the output gate:

$$\vec{c}_t = \vec{i}_t \odot \vec{\tilde{c}}_t + \vec{f}_t \odot \vec{c}_{t-1} \tag{2.32}$$

$$\vec{h}_t = \vec{o}_t \odot \tanh(\vec{c}_t) \tag{2.33}$$

Note that, as the values of $\vec{f}_t$ are derived from $\vec{x}_t$ and $\vec{h}_{t-1}$—and therefore directly *learnable from data*—the LSTM effectively **learns how to appropriately forget**.

2.3.2.4 LSTM regularisation

While the LSTM has effectively surpassed the limitations of SimpleRNNs on many tasks that require capturing long-range temporal dependencies, their success is often condi-

tional on carefully handling their parameters (with appropriate regularisation techniques). The space of such proposed optimisations to LSTMs is substantially large, and I will only present the ones of interest to this dissertation.

Particular care should be exercised when *initialising* the LSTM parameters. The following set of initialisation schemes is often deemed most appropriate, and will be used throughout this document:

- The *recurrent weights*, $\mathbf{U}_*$ are typically initialised as *orthonormal matrices* [153]. This is useful for its properties such as *norm-preservation*, as these matrices will be repeatedly applied to outputs of the recurrent cell.
- The *forget gate bias*, $\vec{b}_f$ is typically initialised as a *ones-vector* [84], i.e. $\vec{b}_f = \vec{1}$. This encourages the network to *"remember"* the values within its memory cell during the early phases of training—which was shown to be critical to appropriately picking up on long-range dependencies later on.
- All other weights are initialised using *Xavier initialisation* [50], as recommended for sigmoidal activations.

Adaptive SGD optimisers such as Adam [88] are still appropriate for training recurrent neural networks. However, as gradient updates are applied many times over on the same recurrent cell, sometimes a less aggressive approach is sought after (especially in contexts where RNNs are applied in conjunction with *reinforcement learning* algorithms). In such situations, the RMSprop optimiser [170] may be used as a controllable alternative.

Techniques such as *dropout* and *batch normalisation* are still appropriate for recurrent neural networks, but special care needs to be taken when applying them as well. Dropout may only be straightforwardly applied to the input-to-hidden transitions within RNNs [196], whereas for *recurrent transitions* the dropout mask needs to be fixed across all timesteps [45]. Similar arguments follow for batch normalisation [28], spearheading the development of specialised operators such as *layer normalisation* [3] and *weight normalisation* [148]. Despite significant strides being made, appropriate normalisation layers for RNNs remain very much an open area of research.

2.3.3 Self-attention

As the final example of simple structural inductive biases, I will direct attention to neural networks optimised to operate over *sets*. Sets represent a simple generalisation of sequences—wherein we don't assume a sequential ordering among the individual steps—and therefore more generic neural network architectures are required for processing them.

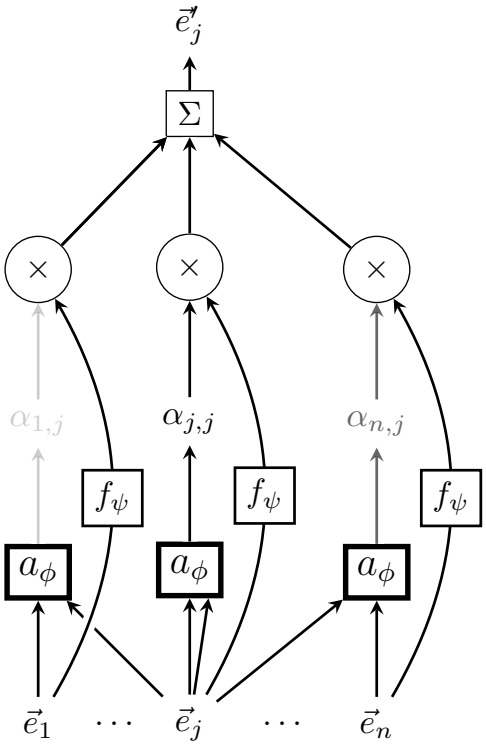


Figure 2.12: The operation of a self-attentional layer (Equations 2.35–2.36) for computing the next-level representation $\vec{e}'_j$ of the j-th element of $\mathcal{E}$.

While several neural network architectures for set-structured data have been recently proposed [179, 193], most prominent directions for handling such inputs leverage *attentional* mechanisms [5]—in particular, the recent developments in **self-attention** [173].

A self-attentional operator, A, acts on an unordered set of n entities[9], $\mathcal{E} = \{\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_n\}$ with the i-th entity described by a *feature vector*, $\vec{e}_i \in \mathbb{R}^m$:

$$\widetilde{\mathcal{E}} = A(\mathcal{E}) \tag{2.34}$$

producing a new set of feature vectors for each entity, $\widetilde{\mathcal{E}} = \{\vec{e}'_1, \vec{e}'_2, \ldots, \vec{e}'_n\}$, $\vec{e}'_i \in \mathbb{R}^k$.

The fact that input sets are now *unordered* introduces a new desirable bias: **permutation invariance**. Permutation invariance implies that, even if we were to *shuffle* the input positions, we would still recover the same result after applying A.

The reason why RNNs are not permutation invariant is that each output vector ($\vec{h}_t$) only considers inputs up to a certain input step. Self-attention extends this concept by allow-

[9] n may be different across different input sets.

ing *each output step to consider **all** input steps*. Namely, each component of $\widetilde{\mathcal{E}}$ will be derived by examining all components of $\mathcal{E}$—by way of bespoke *linear combinations* for each output element (see Figure 2.12):

$$\vec{e}_i' = \sum_j \alpha_{ij} f_\psi(\vec{e}_j) \tag{2.35}$$

Here, $f_\psi : \mathbb{R}^m \to \mathbb{R}^k$ is a learnable transformation (e.g. simple MLP, or even *identity*) with parameters ψ, and α_{ij} is the *attentional coefficient*, specifying the *importance* of entity j's features to entity i. These coefficients are implicitly defined by a shared pairwise function, $a_\phi : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, like so:

$$\alpha_{ij} = a_\phi(\vec{e}_i, \vec{e}_j) \tag{2.36}$$

The function a_ϕ is the **attention mechanism**, and properly choosing it can have substantial effects on the learnt representations. Besides simple deep MLPs [5] or dot product-based approaches [173], more recent interesting work aims to compute similarities between elements in *hyperbolic space* [62], enabling higher representational power for sets exhibiting *hierarchical* properties.

Self-attentional mechanisms have also seen substantial application across tasks where the sets are not entirely unordered, even protruding into the sequential task space—such as machine translation [173]—historically dominated by RNNs. Such operators may be beneficial to recurrent networks, given that all attention computations may be executed *in parallel*, thus requiring $O(1)$ effective depth for a single layer that covers the entire input (as opposed to $O(n)$ for recurrent layers).

2.4 Graph neural networks

A substantial proportion of the work covered in this dissertation concerns neural networks operating on data that are accompanied with some form of *graph structure*. Designing appropriate layers for such inputs is one of the major ongoing challenges of machine learning [7, 16, 65], as graphs present natural generalisations of many popular inputs such as images, text or speech.

2.4.1 Node classification

I will solely focus on the **node classification** task here, as it is the main graph task explored throughout this document. As input, we are provided with a matrix of *node features*, $\mathbf{F} \in \mathbb{R}^{N \times F}$, with F features in each of the N nodes, as well as an *adjacency matrix*, $\mathbf{A} \in \mathbb{R}^{N \times N}$. The objective is to *classify* each of the nodes into one of C classes, i.e. to pro-

duce a matrix of *node class probabilities*, $\mathbf{Y} \in \mathbb{R}^{N \times C}$, such that $Y_{ij} = \mathbb{P}(\text{Node } i \in \text{Class } j)$.

For simplicity, I will also consistently assume that the graphs are **unweighted** and **undirected**, i.e. that $\mathbf{A}$ is symmetric and binary:

$$A_{ij} = A_{ji} = \begin{cases} 1 & i \leftrightarrow j \\ 0 & \text{otherwise} \end{cases} \tag{2.37}$$

However, most of the algorithms discussed here are capable of generalising to more generic kinds of edges—even ones exhibiting arbitrary *vector-valued features*.

In this space, there are two main kinds of learning tasks:

- *Transductive* learning—wherein the training algorithm has access to *the entirety of the input graph's features*, **including the test nodes**. This learning setup may be seen as a semi-supervised task of *propagating labels* from training nodes to the remainder of the graph.
- *Inductive* learning—here, the algorithm will not have access to all nodes upfront. This implies that, either, we are dealing with an *evolving* graph wherein test nodes are *incrementally added* or, more generally, there exist **disjoint** and *unseen* test graphs. This is a substantially harder learning problem, which requires generalising across *arbitrary graph structures*, and many transductive graph learning algorithms will be theoretically inappropriate in the inductive setting.

Clearly, the simplest way of approaching this task is to *drop the graph structure entirely*, and simply have a shared per-node classifier (e.g. an MLP). In fact, this setup corresponds to the *majority* of deep learning applications nowadays—given that, in many cases, graph structure can be derived between individual training examples in a dataset[10]. I will present two approaches—the main focus of this dissertation's contributions—that go beyond this simple classifier to *incorporate* graph structure rather than drop it.

2.4.2 Random walk-based node embeddings

One such approach *decouples* the incorporation of the graph structure from the processing of input node features[11]. Namely, for each node i, *structural features*, $\vec{\Phi}_i$, are derived, considering only the adjacency matrix information. Afterwards, the concatenation of

[10] However, a graph-based network requires simultaneously storing all training nodes in GPU memory. Therefore, such approaches are often prohibitively memory-intensive on high-dimensional datasets.
[11] This simplifies the processing to an extent when input graphs are *featureless*.

structural features with the input features, $\vec{f}_i \| \vec{\Phi}_i$, can be used as input to a shared per-node classifier, just as before.

Typically, **random walks** in a graph are used as the primary proxy for analysing its structural information, and the first method to successfully exploit this information with deep neural networks is *DeepWalk* [131]. As most random-walk based methods that came after it, such as *node2vec* [61] and *LINE* [169], primarily focus on modifying the way in which the random walks are constructed without modifying the primary idea, in the remainder of this section I will focus solely on a brief exposition of DeepWalk.

DeepWalk draws inspiration from **skip-gram** methods in natural language processing [116], generalising them to graphs by treating random walks as *sentences* and individual nodes as *words*—a "good" structural node representation should be *predictive* of the nodes *surrounding* it (i.e. nodes that occur close to it in a random walk).

Specifically, the algorithm initially stores *random* structural features $\vec{\Phi}_i$ for each node i. At each step, a random walk, $\mathcal{W}$, following along the edges of the graph, is sampled. Considering the node at its j-th step, $x = \mathcal{W}_j$, and a node at its k-th step, where $k \in [j - w, j + w]$ (where w is an appropriately chosen *window size*), $y = \mathcal{W}_k$, the algorithm modifies $\vec{\Phi}_x$ to maximise $\log \mathbb{P}(y|\vec{\Phi}_x)$—obtained from a neural network-based classifier. The entire architecture is trained end-to-end using gradient descent.

It should be noted that implementing the prediction network as a simple neural network classifier is *prohibitive* for large graphs, given that it features an N-way softmax across all the nodes—making most probabilities *negligibly small* early on, and therefore making most gradient updates *vanish*. DeepWalk rectifies this issue through a *hierarchical softmax* layer (representing the prediction as a *tree* of binary classifiers, each halving the space of consdered nodes), while most of the more recent techniques tend to incorporate *negative sampling* as an alternative.

Methods such as DeepWalk are still favourable when dealing with *fully unsupervised* graph tasks, as they don't depend on having labels or features in the nodes. However, if this information is available, it is often quite beneficial to use it—leading to random-walk based *supervised* methods such as *Planetoid* (Predicting Labels And Neighbours with Embeddings Transductively Or Inductively from Data) [190].

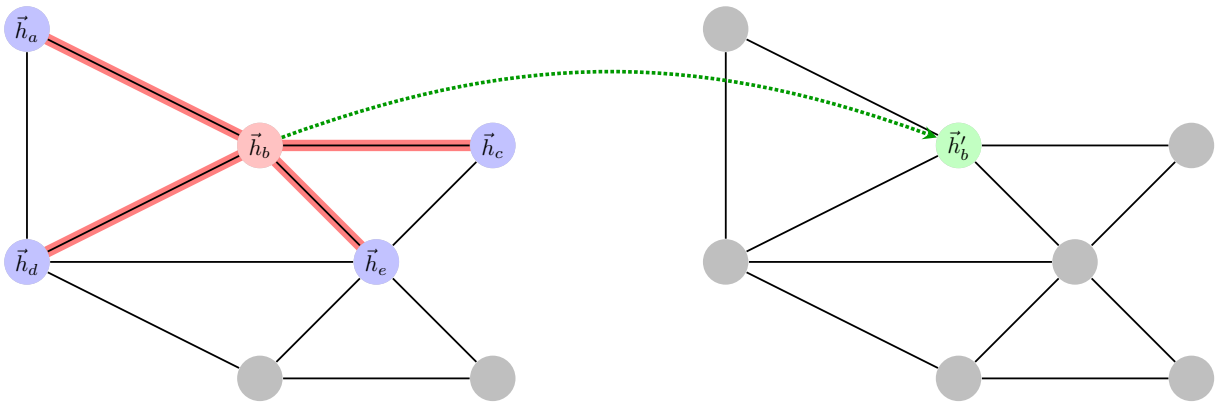


Figure 2.13: In the most usual formulation, a graph convolutional layer derives next-level features for each node (e.g. $\vec{h}'_b$ in this case) by taking into account all nodes within its (usually first-order) neighborhood. These features, therefore, provide a summary of *patches* centered around each node.

2.4.3 Graph convolutional networks

Conversely, this subsection will consider techniques that leverage the graph structure *directly* while extracting intermediate feature representations, $\vec{h}_i$, for each node i in the graph—these representations may then be used to classify each node separately, as before. Dependent on context, these techniques may be referred to as *graph neural networks* [58, 107, 154] or *graph convolutional networks* [17, 33, 92].

Hereinafter, I will assume the term *graph convolutional networks* to refer to these techniques, as it nicely relates them to a generalisation of the *convolutional layer* from CNNs to graph-based neural networks. It should be noted that, in general, all of the architectures to be presented in this document can be reformulated as a particular instance of *message-passing neural networks* [49]. In particular, I will present the *graph convolutional network* (**GCN**) of Kipf and Welling [92], which is the most popular such layer at the time of writing this dissertation.

Akin to convolutional layers within CNNs, a *graph convolutional layer* (Figure 2.13), g, obtains higher-level representations of a node i by leveraging the information in its *neighbourhood*, $\mathcal{N}_i$ (often including the node itself):

$$\vec{h}'_i = g(\vec{h}_a, \vec{h}_b, \vec{h}_c, \dots) \quad (a, b, c, \dots \in \mathcal{N}_i) \tag{2.38}$$

Images have a highly rigid and regular connectivity pattern, making convolutional layers trivial to deploy (as a small kernel matrix which is slided across). Generalising this to arbitrary graphs, however, represents a **much harder** challenge.

One simple way to aggregate node neighbourhoods is by multiplying the node feature matrix with the adjacency matrix:

$$\mathbf{H}' = \sigma\left(\mathbf{AHW}\right) \tag{2.39}$$

where $\mathbf{W}$ is a shared, node-wise, learnable linear transformation (necessary in order to derive higher-level features), and σ is a nonlinearity.

A few additional aspects of this layer need to be fixed in order to make it viable. Initially, it *discards* the central node, which can result in a catastrophic loss of information. A simple correction with inserted self-loops fixes this problem:

$$\mathbf{H}' = \sigma\left(\tilde{\mathbf{A}}\mathbf{HW}\right) \tag{2.40}$$

Here, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is the adjacency matrix with inserted self-loops. Furthermore, the multiplication by $\tilde{\mathbf{A}}$ may modify the *scale* of the output features, and therefore needs to be *normalised* appropriately, e.g. by applying the following modification:

$$\mathbf{H}' = \sigma\left(\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}\mathbf{HW}\right) \tag{2.41}$$

where $\tilde{\mathbf{D}}$ is the *degree matrix* of $\tilde{\mathbf{A}}$, i.e. $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, and it is zero off-diagonal. This fully specifies the *mean-pooling* update rule, written out node-wise as follows:

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \frac{1}{|\mathcal{N}_i|}\mathbf{W}\vec{h}_j\right) \tag{2.42}$$

which is a simple but *versatile* operator, applicable in inductive settings as well (as the information on the neighbourhood size can be obtained by simple local message passing).

The **GCN** update rule is obtained by, instead, using *symmetric normalisation*:

$$\mathbf{H}' = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{HW}\right) \tag{2.43}$$

which, written out node-wise, is as follows:

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}}\mathbf{W}\vec{h}_j\right) \tag{2.44}$$

The use of symmetric normalisation allows for emergence of more interesting dynamics, and can be related to a particular approximation of applying the convolution theorem

to signals defined on graphs [17, 33]. This causes the layer to be simple, powerful and robust to overfitting—however, it is in theory not applicable to inductive problems, as knowledge of neighbourhood sizes implies that the learnt filters rely on knowing the entire graph structure upfront.

Chapter 3

Cross-modality through early fusion

> "Things are *sometimes* fun, when you know *exactly* what you know."
>
> Mara Mihali

Inherently, for most machine learning tasks of interest, we will have access to a multitude of input data sources; i.e. we will be performing *multimodal* machine learning [124]. Appropriately exploiting this wealth of information remains a key challenge, especially when the corresponding training set sizes are small—i.e. the *wide data*, or "small n, large p" regime—which is extremely common in the biomedical space.

Taking into account that, in many cases of interest, the different input modalities will also come with *different structural biases* (for example, we may be asked to combine audiovisual information), it is often assumed that each modality will be *independently* processed by a neural network *specialised* for handling its biases. Each of these networks will extract a set of *high-level features* for its modality, which are then concatenated across all modalities for the purposes of prediction tasks.

This is the *late fusion* approach [2, 85], which is often simple to implement but potentially misses out on opportunities to augment the individual feature extractors with useful *cross-modal information*—and many multimodal signals exhibit correlations that should be exploitable even on this level. The alternative approach—*early fusion*—is often neglected given its requirement for bespoke methods of cross-modal communication.

As an important step towards viable early fusion methods, in this chapter I restrict myself to the case where the structural inductive biases are identical across modalities (i.e. the modalities are either *all grid-like* or *all sequential*). This simplifies the design of the cross-communication architecture between the feature extractors, allowing for focus on evalu-

ating the benefits of performing early fusion. I successfully demonstrate that early fusion provides tangible benefits, within both CNN and LSTM architectures—especially in settings when the sample sizes are small and/or when important features remain unobserved. Furthermore, despite the fact that the cross-modal architecture requires specification of a substantial further number of hyperparameters, I show that they can be optimised in a principled manner, requiring careful tuning of only a *single parameter*.

Lastly, I indicate further work I have been involved in that expanded these early fusion ideas into the domain of generic structural inductive biases for each modality—primarily, by demonstrating a successful early fusion system for audiovisual data.

3.1 Methodology

The early fusion methodology depicted here is inspired by *multilayer networks* [94], mathematical structures encompassing several layers of graphs over the same set of nodes, allowing for unrestricted intra-layer as well as inter-layer connections. They have been a demonstrably valuable tool for modelling a variety of natural and social systems [31, 43, 59], and their applicability to machine learning was already demonstrated within the context of hidden Markov models [178], managing to achieve high performance on a sparse breast cancer classification dataset involving gene expression and methylation data.

The network design process is initiated by appropriately partitioning the input data—this may be done either manually (by exploiting existing domain knowledge) or through an unsupervised pre-training step that will determine which (not necessarily disjoint) fragments of the input data are more likely to constructively influence one another. Afterwards, a cross-modal network is constructed such that a separate neural network *superlayer* (CNN or LSTM) is dedicated to each partition of the input data. The purpose of the partitioning is to help the constituent networks become powerful predictors while requiring a smaller dimensionality of the input data, by allowing them access to those parts of the input which are most significantly related to each other in the context of the predictions that need to be made.

Finally, the superlayers may be interconnected by any (feedforward) *cross-connection* as is best seen fit, and they may be combined in arbitrary ways at the output stage to produce the final output. This construction is biologically inspired by *cross-modal systems* [40] within the visual and auditory systems of the human brain (which in turn inspired the development of CNNs)—wherein several cross-connections between various sensory networks have been discovered [8, 189].

I will now outline the ways in which cross-connections may be constructed, separately handling the case of *cross-modal convolutional neural networks* (**X-CNN**s) [176] and *cross-modal long short-term memories* (**X-LSTM**s) [177]. Lastly, I will highlight a simple but effective method for performing hyperparameter tuning on such models [87].

3.1.1 Cross-connections

Assume, for simplicity, that we have only two modalities, with (intermediate) feature representations $\vec{x}_a$ and $\vec{x}_b$ that we wish to cross-connect. In this case, we initially compute *intra-layer* and *inter-layer* feature representations, and then *recombine* them into outputs ($\vec{y}_a$ and $\vec{y}_b$) as follows:

$$\vec{h}_{a\to a} = \phi_a(\vec{x}_a) \qquad \vec{h}_{b\to b} = \phi_b(\vec{x}_b) \tag{3.1}$$

$$\vec{h}_{a\rightsquigarrow b} = \chi_{ab}(\vec{x}_a) \qquad \vec{h}_{b\rightsquigarrow a} = \chi_{ba}(\vec{x}_b) \tag{3.2}$$

$$\vec{y}_a = \gamma_a(\vec{h}_{a\to a}, \vec{h}_{b\rightsquigarrow a}) \qquad \vec{y}_b = \gamma_b(\vec{h}_{b\to b}, \vec{h}_{a\rightsquigarrow b}) \tag{3.3}$$

Here, $\phi_\cdot$, $\chi_\cdot$ and $\gamma_\cdot$ are (potentially learnable) feature transformations—often simple neural network layers, chosen appropriately to respect the structural biases between each of the inputs. The scheme easily generalises to more than two modalities, and it is also possible to omit computing features in certain directions. For all cross-modal instances considered here, the recombination function γ will be *featurewise concatenation* (i.e. $\gamma(\vec{a}, \vec{b}) = \vec{a} \| \vec{b}$), allowing each superlayer maximal freedom with how it chooses to process the incoming information. This effectively preserves the dataflow benefits of a *fully connected* neural network, while requiring substantially fewer parameters; as such, it is expected that such architectures will generalise better under challenging data scenarios (e.g. datasets with *small sample sizes* or *missing* observations).

3.1.2 X-CNN

Here I consider the case of multimodal input wherein each modality constitutes an *image* of the same width and height. While somewhat specific, this setup still holds high practical applicability—e.g. in *medical imaging*, we may think of individual images as representing a patient's medical scans (CT/MRI/PET), allowing insights into different granularities of anatomical organisation. The individual images may also be obtained from a single base image, by e.g. decoupling its individual channels (in either YUV or RGB space).

In this case, a separate CNN feature extractor is utilised for each of the images. Early

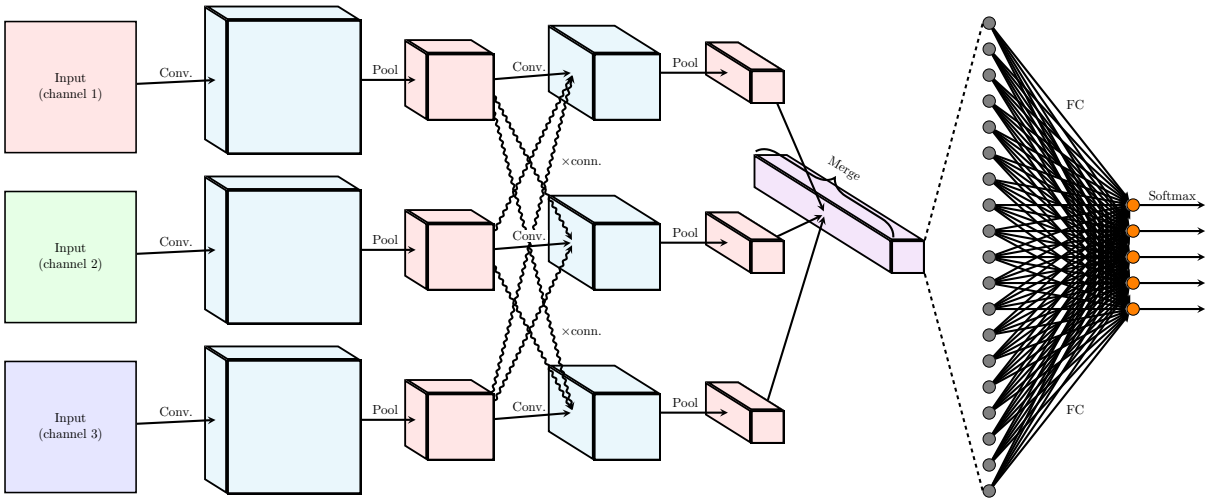


Figure 3.1: Diagram of a simple cross-modal CNN for image classification, generated from a baseline CNN of the form [Conv $\rightarrow$ Pool] $\times$ 2 $\rightarrow$ FC $\rightarrow$ Softmax. Each of the three channels (e.g. RGB/YUV) of the input image receives its own CNN superlayer, with cross-connections inserted after the pooling operation. A more in-depth view of a potential cross-connection layout is provided by Figure 3.2.

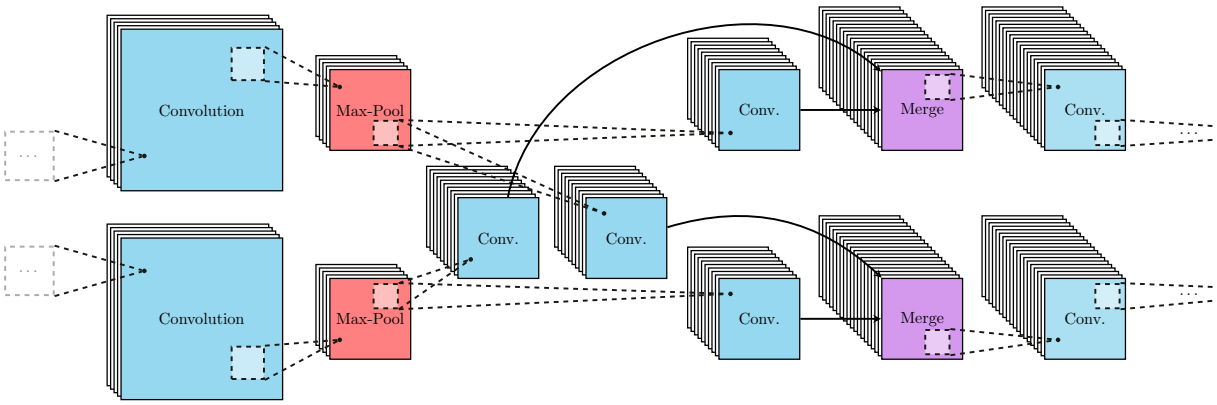


Figure 3.2: Illustration of a single cross-connection segment within an X-CNN with two superlayers. After each pooling operation, feature maps between the superlayers are exchanged, after first passing them through an additional convolutional layer. We may also perform an additional intra-superlayer convolution before merging the feature maps in each superlayer via concatenation.

fusion is performed by way of *convolutional cross-connections*, allowing the individual extractors to exchange information, as in Figure 3.1. The exact implementation of the cross-connection can vary; here it constitutes *feature map exchange*, i.e. it allows each of the feature extractors to receive intermediate feature maps from other extractors throughout the pipeline. Therefore, the functions $\phi_\cdot$ and $\chi_\cdot$ (of Equations 3.1–3.3) are convolutional layers, and $\gamma_\cdot$ is channel-wise concatenation.

Within the specific implementation attempted in [176], I have applied 1×1 convolutions to transform the feature maps from one CNN stream before passing them to another (as in Figure 3.2). The motivation between such an operator is that it learns a *pointwise transformation of pixels*, such that they become more usable for the other stream with minimal parameter usage. This kind of cross-connection was concurrently developed (under the name *lateral connection*) by [145], where it was used for enabling information exchange between task-specialised neural networks in *continual learning*.

The cross-connections are placed at the end of every *pooling* (downsampling) layer of the individual CNNs—with the motivation that the decrease in width and height provides additional space for augmenting the *channels* with the newly received feature maps.

3.1.3 X-LSTM

Now consider the multimodal deep learning setting where the input modalities are *aligned* sequences. Once again, while potentially specific, such settings naturally correspond to several measurements taken at regular intervals (e.g. from various sensors), and many irregular setups can be transformed to it through methods such as *interpolation*.

Here a natural choice for each of the superlayers is a recurrent neural network (such as an LSTM). I have investigated several candidates for early fusion, finding *recurrent cross-connections* to perform the best. In this case, intermediate time-series features are exchanged between superlayers by first passing them through another recurrent layer, followed by concatenation at each timestep. As such, the functions $\phi_\cdot$ and $\chi_\cdot$ of Equations 3.1–3.3 are LSTM layers, and $\gamma_\cdot$ is featurewise concatenation.

Unlike the X-CNN setup, here the "depth" of the LSTMs typically will not exceed three layers. As such, only a single cross-connection is placed (within the second layer), and the third layer is used for intra-layer summarisation (as depicted in Figure 3.3).

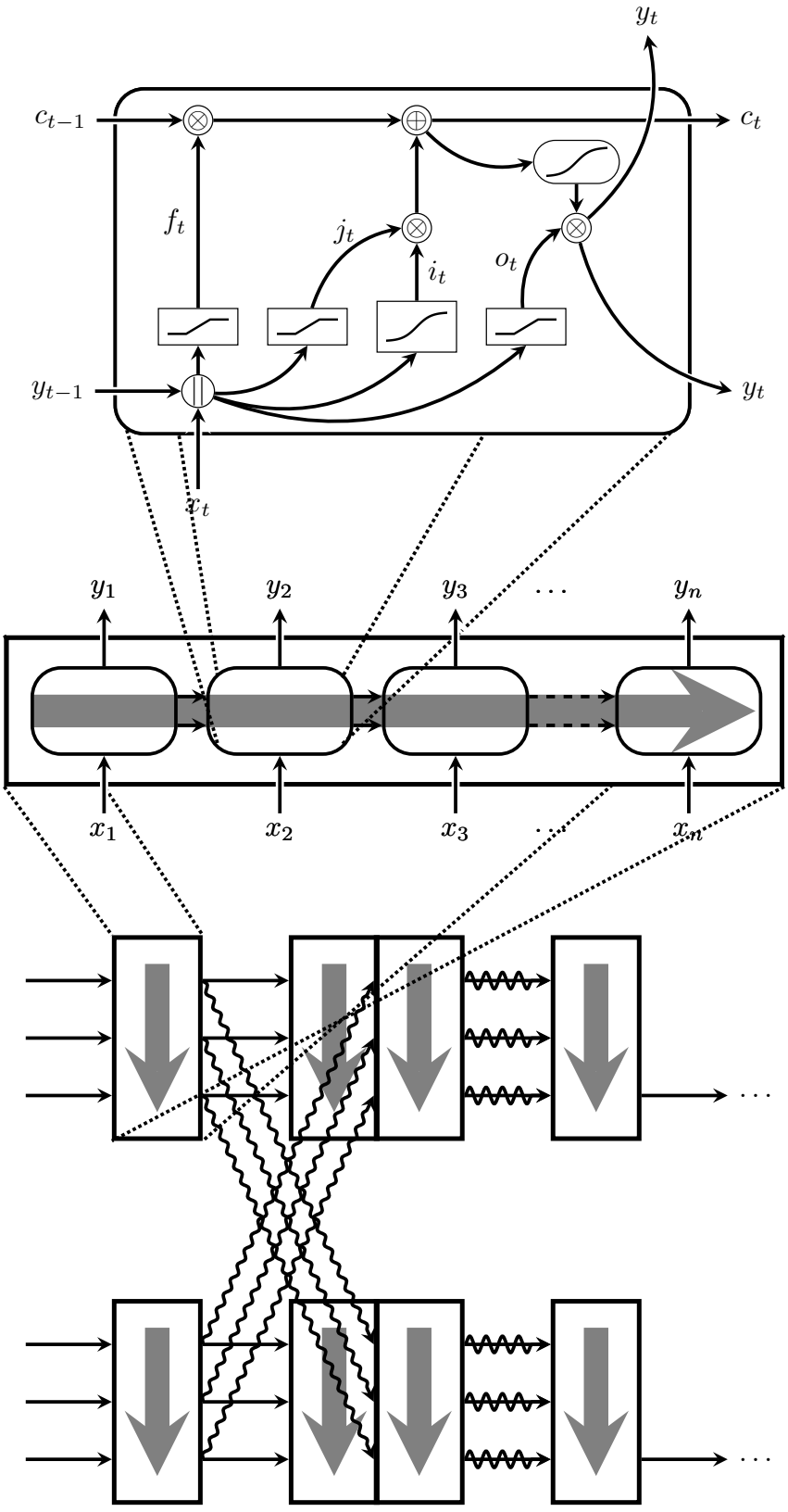


Figure 3.3: A hierarchical diagram of a 3-layer X-LSTM model with one cross-connection in the second layer. **Top:** A single LSTM cell. **Middle:** An LSTM layer (replicated cell). **Bottom:** A 3-layer cross-modal LSTM model with 2 superlayers. In the second layer, the hidden sequences are passed through a separate LSTM layer and featurewise concatenated with the main stream sequence to facilitate sharing.

3.1.4 Hyperparameter tuning

Despite the clear potential of cross-connected architectures, they also carry an obvious and significant **overhead**—namely, as we need to specify how *each pair* of superlayers will interact when cross-connected, the number of design decisions now grows at least *quadratically* with the number of modalities available. This makes such models potentially unsuitable for widespread application in the general case.

It turns out, however, that effective hyperparameter optimisation is possible for both the X-CNN and X-LSTM, and it often can be reduced to optimising a *single* hyperparameter. The technique relies on leveraging *unimodal performance metrics* to inform the model on how many features should be dedicated to each modality, as well as sent across when cross-connecting.

Upon first presenting X-LSTMs [177], I have devised a simple hand-crafted version of this technique, which proceeds as follows:

- First, construct a *unimodal* LSTM, which processes only a single modality.
- Evaluate the performance of this LSTM when applied to each modality in turn, e.g. on a held-out validation set, obtaining scores s_m (e.g. accuracy, area under ROC curve, etc.) for each modality m.
- When constructing the X-LSTM, upon deciding how many features to allocate to each modality's superlayer, construct them such that they respect the *ratio* of these scores, e.g. for three modalities a, b and c, the superlayers' feature counts should respect the ratio $s_a : s_b : s_c$. Similarly, the decision on the number of features sent across for a cross-connection of type $a \rightsquigarrow b$ should respect the ratio $s_a : s_b$.
- In many cases, the unimodal performance metric will be too similar, and therefore respecting the direct score ratio may cause the produced networks to be "too uniform". As such, discrepancies can be enforced by raising the scores to a power parameter, k. The feature counts should then be chosen to respect the ratio $s_a^k : s_b^k : s_c^k$.

Thus, assuming a desirable overall parameter count, specifying the hyperparameter k can fully specify the choice of feature counts. Empirically, this approach has lead to highly desirable architecture configurations with minimal tuning effort.

3.1.5 Xsertion

Subsequently, this intuitive approach was formalised into the **Xsertion** library [87] by Laurynas Karazija, for his Part III project performed under my co-supervision. Xsertion

consumes as input a *dataset* (with a specified training/validation split) and a *baseline CNN*, producing a *hyperparameter-tuned X-CNN* with a comparable parameter count to the baseline. Given a unimodal score, n_{l_i}, for modality i, the scaling multiplier for the i^{th} superlayer's feature counts is defined as follows:

$$s_{l_1} = \frac{n_{l_1}^{\alpha}}{\sum_i n_{l_i}^{\alpha}}. \tag{3.4}$$

where α is a hyperparameter used for tuning the prioritisation of higher informativeness (equivalent of k from before).

Similarly, cross-connection feature counts are optimised through a scaling multiplier called the *connection weight*. A desired weight $w_{l_1,l_2} \in [0, 1]$ of a connection between superlayers $l_1 \rightsquigarrow l_2$ is such that $w_{l_1,l_2} > 0.5$ when connecting a node from a more informative position to a less informative one. This can be parametrised as:

$$w_{l_1,l_2} = \frac{n_{l_1}^{\beta}}{n_{l_1}^{\beta} + n_{l_2}^{\beta}}, \tag{3.5}$$

where β is a hyperparameter controlling the discounting of lower informativeness. With $\beta \to 0$, all modality pairs are treated equally. When $\beta \to \infty$, most of the weight is assigned to the features transferred from the more informative superlayer. If a weight is set to zero, the corresponding cross-connection is dropped.

Besides determining the optimal feature counts within cross-connections (which effectively reduces to appropriately tuning α and β), Xsertion also optimises for the *locations* where cross-connections should be placed, which is another potentially challenging design decision. For purposes of brevity, I omit the discussion of such aspects of Xsertion, but it should be noted that, taken in unison, they allow the neural network designer to be extremely lenient when designing a X-CNN architecture.

3.2 Experimental setup

Having described the essential components of all the considered cross-modal architectures, in this section I will outline the experimental procedure used to evaluate the relative benefits of X-CNNs and X-LSTMs against baseline approaches, highlighting in particular the chosen datasets, architectural hyperparameters and the exact learning and evaluation setup used to obtain the final performance comparisons.

3.2.1 Datasets and preprocessing

As was previously mentioned, it is especially assumed that the outlined early fusion architectures will perform well in challenging data scenarios. For the case of X-CNNs, it is possible to leverage large-scale *image datasets*, to more directly quantify these benefits (by *holding out* increasing quantities of training images from the learning algorithm).

Consequently, for evaluating the X-CNN construction I leverage the standard CIFAR-10 and CIFAR-100 datasets [97], involving classifying tiny coloured images ($32 \times 32 \times 3$) into 10 and 100 classes, respectively. For both of these datasets, an abundance of data is available (50,000 training and 10,000 testing examples). This makes it easier to study the behaviour of the models as different fractions of the training data are discarded. I hypothesise that, at lower levels of data availability (up to a *threshold*), X-CNNs will yield significant gains over equivalent unrestricted CNNs—and also that they will remain competitive at all higher training set sizes.

At all times, images are represented in the **YUV** colour space. As a linear transformation from RGB, it should not have an impact on performance of the baselines, while it has the benefit of decoupling *luminance* from *chrominance*, allowing for a simpler analysis of cross-connections (and relating its learned kernels to human vision processes). The images are preprocessed by applying a single *batch normalisation* operation on them; I have found this to yield slightly better results compared to doing global contrast normalisation and ZCA whitening.

Multimodal sequential datasets of similar properties (to be used for evaluation of X-LSTMs) are harder to come by. For example, the majority of open datasets in such spaces consist of *textual* data, for which each step of the sequence will consist of a *word embedding*—the embeddings being arbitrary vectors, they may not be trivially split into modalities as is the case with image channels.

Recently, consumer-grade health devices, such as wearables and smart home appliances became more widespread, which presents new data modelling opportunities. For the purposes of evaluating the X-LSTM, I investigate one such task—predicting the users' future body weight in relation to their weight goal given historical weight, along with sleep and steps measurements [177]. This study is enabled by a first-of-its-kind dataset of fitness measurements from *∼15,000 users*. Data are captured from various sources, such as smartwatches, wrist- and hip-mounted wearables, smartphone applications and smart bathroom scales. This is one of the first times that such quantities of large-scale longitudinal (spanning *up to 500 consecutive days* of comprehensive measurements recorded per

user) multi-device consumer-grade health data have been investigated. From this dataset, a binary classification task is studied: *will the users be able to achieve the weight goal they input into their smart device?*

Besides the inherent multimodality and sequentiality of the input data, this dataset also poses a further modelling challenge which makes it suitable for evaluating the benefits of early fusion. Namely, modelling even thousands of users limits the kind of data that sufficiently many user devices can accurately measure. Therefore, many factors key in weight change (such as *eating habits*) must remain as only *latently* observed.

There is a range of potential scenarios where such predictive modelling would be useful for weight control within consumer health systems. Motivating examples include: *direct feedback to the user about their progress*, *suggesting new, realistic weight objectives*, and *evaluating the effects of major lifestyle changes*. Significant work already exists towards developing consumer systems of this type [99, 106, 111, 184] but they often assume the availability of scalable predictive models of user behaviour, such as the weight goal prediction task I investigate.

This investigation was performed on anonymised data obtained from several devices across the *Nokia Digital Health - Withings*[1] range. The dataset contains weight, height, sleep and steps measurements, as well as user specified weight objectives. Weights are measured by the Withings scale. All other data are obtained from the Withings application through the use of wearables.

Users were first included in the dataset under the condition of having recorded at least 10 weight measurements over a 2-month period. In total, the dataset contains 1,664,877 such users. Further processing was performed to remove outliers or those users with too few, or too sporadic, data observations; after this stage $\sim$15K users were remaining. The precise steps taken to reach this final dataset are described below.

Obvious outliers, reports of unrealistic heights (below 130 cm or above 225cm), and/or consistent weight changes of more than 1.5kg per day have been discarded. Steps and sleep are recorded on a per-day basis, while weights are recorded at the user's discretion; to align the weight measurements with the other two modalities, we have applied a moving average to the person's recorded weight throughout an individual day. A sequence may be labelled with any weight objective that has been set by the user, and is still unachieved, by the time the sequence ends. Overly ambitious objectives (over $\pm$20

[1] At the time of writing this dissertation, known only as *Withings*.

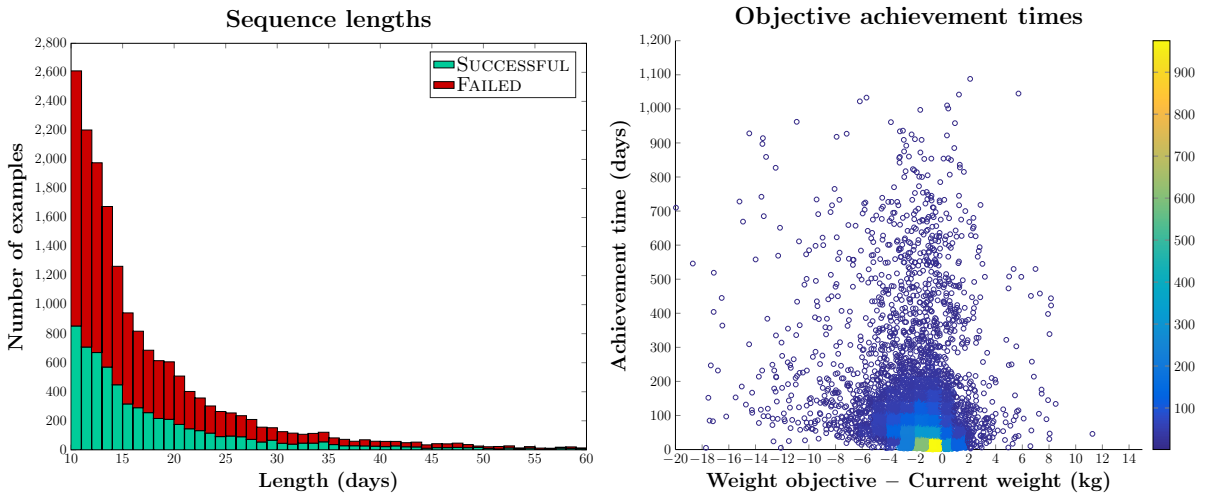


Figure 3.4: **Left:** Plot of the sequence length distribution in the final dataset. **Right:** Mixed heatmap/scatter plot of the weight objectives against their achievement times, for the successful sequences in the final dataset.

kilograms proposed) are ignored.

A weight objective is considered to be *successful* if there exists a weight measurement in the future that reaches or exceeds it, and unsuccessful if the user *stops recording weights* (allowing for a long enough window after the end of the recorded sequence) or *sets a more conservative objective in the meantime*. The derived dataset spans 18,036 sequences associated with weight objectives. Within it, 6,313 of the sequences represent successful examples, while the remaining 11,723 represent examples of failure. To address the potential issues of class imbalance, appropriate class weights are applied to all relevant optimisation targets and loss functions.

All of the sequences are comprised of user-related features: height, gender, age category, weight objective; along with sequential features—for each day: duration of light and deep sleep, time to fall asleep and time spent awake; number of times awoken during the night; time required to wake up; bed-in/bed-out times; steps and (average) weights for the day. Only sequences that span at least 10 contiguous days are considered. There are ∼3.6K users for whom more than one sequence is present in the dataset. In line with known best practices in deep learning, data are normalised to have mean zero and standard deviation one per-feature.

In order to get an impression of the statistics present within the dataset, we have visualised the sequence length distributions (outliers removed for visibility, but still used in our analysis), as well as scatter plots of successful weight objective magnitudes against their achievement times. These are provided by Figure 3.4. The median time of achieve-

ment for successfully achieved objectives is 57 days.

3.2.2 Model architectures

For purposes of reproducibility, in this section I will expose the architectural hyperparameters of the models used for evaluation. The cross-modal architectures' feature map counts have been altered in such a way as to make the overall number of parameters as close as possible to their baseline. This ensures as-fair-as-possible comparison with respect to degrees-of-freedom.

For evaluating the X-CNN construction, the investigation spans *two* models:

- A simple CNN with four convolutional layers, followed by two fully connected layers. I will be referring to it as *KerasNet* throughout this chapter as it is based on the Keras CIFAR-10 CNN example [25]. It represents a likely style of a "starting" model that one is going to attempt to apply on an image classification problem. The architecture of the model, as well as its cross-modal variant (X-KerasNet) is outlined in Table 3.1. It is trained for 200 epochs using the Adam SGD optimiser, with hyperparameters as described in [88], and a batch size of 32. Dropout has been applied after both pooling operations (with $p = 0.25$) as well as after the first fully connected layer (with $p = 0.5$).

- The *FitNet4* architecture [139], representing a sophisticated CNN close to the state-of-the-art on CIFAR-10/100. This model was chosen for its prominent appearance in topical CNN research [117, 164], and due to its design goal of being a "thin&deep" network, managing to keep its parameter count relatively low compared to many other successful models. The FitNet4 consists of 17 convolutional 2-way maxout [56] layers, followed by two fully connected layers, the first of which is a 5-way maxout layer. The full architecture of this model—as well as its cross-modal variant (X-FitNet4)—is presented in Table 3.2. The models are initialised using Xavier initialisation [50], and are then trained for 230 epochs using the Adam SGD optimiser with a batch size of 128. Batch normalisation [78] has been applied to the output of each hidden layer to significantly accelerate the training procedure. L_2 regularisation with $\lambda = 0.0005$ has been applied to all weights in the model. Finally, dropout (with $p = 0.2$) was applied on the input, after every pooling operation, and after the fully connected maxout layer.

It should be noted that further domain knowledge is injected into these hand-crafted X-CNN models by favouring the CNN superlayer corresponding to the Y channel in terms of feature map counts (typically doubled compared to the U/V superlayers within the

Table 3.1: Architectures for KerasNet and X-KerasNet

Output size	KerasNet ~ 4.46M param.	X-KerasNet ~ 4.37M param.
32×32	$[3 \times 3, 64] \times 2$	Y: $[3 \times 3, 32] \times 2$
		U/V: $[3 \times 3, 16] \times 2$
16×16	2×2 Max-Pool, stride 2	
		Y $\rightarrow$ Y: *identity*
		U $\rightarrow$ U: *identity*
		V $\rightarrow$ V: *identity*
		Y $\rightsquigarrow$ U/V: $[1 \times 1, 32]$
		U/V $\rightsquigarrow$ Y: $[1 \times 1, 16]$
	$[3 \times 3, 128] \times 2$	Y: $[3 \times 3, 64] \times 2$
		U/V: $[3 \times 3, 32] \times 2$
8×8	2×2 Max-Pool, stride 2	
1×1	Fully connected, 512-D	
	10/100-way softmax	

Table 3.2: Architectures for FitNet4 and X-FitNet4

Output size	FitNet4 ~ 2.75M param.	X-FitNet4 ~ 2.72M param.
32×32	$[3 \times 3, 32] \times 3$	Y: $[3 \times 3, 24] \times 3$
		U/V: $[3 \times 3, 12] \times 3$
	$[3 \times 3, 48] \times 2$	Y: $[3 \times 3, 36] \times 2$
		U/V: $[3 \times 3, 18] \times 2$
16×16	2×2 Max-Pool, stride 2	
		Y $\rightarrow$ Y: $[1 \times 1, 36]$
		U $\rightarrow$ U: $[1 \times 1, 18]$
		V $\rightarrow$ V: $[1 \times 1, 18]$
		Y $\rightsquigarrow$ U/V: $[1 \times 1, 12]$
		U/V $\rightsquigarrow$ Y: $[1 \times 1, 12]$
	$[3 \times 3, 80] \times 6$	Y: $[3 \times 3, 60] \times 6$
		U/V: $[3 \times 3, 30] \times 6$
8×8	2×2 Max-Pool, stride 2	
		Y $\rightarrow$ Y: $[1 \times 1, 60]$
		U $\rightarrow$ U: $[1 \times 1, 30]$
		V $\rightarrow$ V: $[1 \times 1, 30]$
		Y $\rightsquigarrow$ U/V: $[1 \times 1, 18]$
		U/V $\rightsquigarrow$ Y: $[1 \times 1, 18]$
	$[3 \times 3, 128] \times 6$	Y: $[3 \times 3, 96] \times 6$
		U/V: $[3 \times 3, 48] \times 6$
1×1	8×8 (global) Max-Pool	
	Fully connected, 500-D	
	10/100-way softmax	

same hidden layer). This corresponds to the assumption that the majority of relevant information about an object is contained within its brightness channel, while colour usually represents auxiliary information. For both baselines, I will also report the scores obtained by applying the Xsertion library [87].

For evaluating X-LSTMs, the primary baseline architecture represents a three-layer deep LSTM model for processing the historical weight/sleep/steps data. After performing the LSTM operations, the features of the final computed LSTM output step are concatenated with the user's height, gender, age category and weight objective for downstream prediction. As previously discussed, an X-LSTM variant would only have a cross-connection in the second layer.

To construct competitive X-LSTMs, I have applied the manual hyperparameter optimisation technique detailed in Section 3.1.4. Therefore, I have computed the AUCs of the individual unimodal LSTMs on a validation dataset, obtaining AUCs of 80.62% (for weight), 80.17% (for sleep) and 74.18% (for steps). As the scores were not disparate enough in order to generate non-uniform X-LSTMs, I have performed a grid search on the power parameter k. The X-LSTM performed the best with $k = 30$. Its architecture is reported in Table 3.3. Furthermore, note the slightly surprising result: **sleep** being almost *equally* predictive of future weight change as **weight**. This result will be compounded by the qualitative analysis of the model behaviour in Section 3.3.2.

For both the LSTM and X-LSTM, the fully connected layers of the networks apply ReLU activations. The LSTM weights are initialised with Xavier initialisation [50], and its forget gate biases with ones [84]. Finally, the fully connected weights are initialised using He initialisation [67]. The models are trained for 200 epochs using the Adam SGD optimiser, with hyperparameters as described in [88], and a batch size of 1024. For regularisation purposes, batch normalisation is applied to the output of every hidden layer, as well as dropout (with $p = 0.1$) to the input-to-hidden transitions within the LSTMs [196].

3.2.3 Evaluation protocol

Lastly, I will present the evaluation protocols for verifying the benefits of the proposed cross-modal architectures in challenging data scenarios.

Evaluating the validity of the claim about X-CNN performance is performed, as previously discussed, by performing comparative evaluation on CIFAR-10/100. In each individual test, the accuracy of the considered models is evaluated on the entire test set of 10,000 samples, when the training routine is presented with only p% of the entire train-

Table 3.3: LSTM and X-LSTM model architectures. Cross-connections are **highlighted**.

LSTM	X-LSTM ($k = 30$)
76377 param.	75089 param.
21 features	wt: 15 features, sl: 12 features, st: 2 features
	wt $\rightsquigarrow$ sl: 9 features, wt $\rightsquigarrow$ st: 14 features
	sl $\rightsquigarrow$ wt: 6 features, sl $\rightsquigarrow$ st: 11 features
	st $\rightsquigarrow$ wt: 1 feature, st $\rightsquigarrow$ sl: 1 feature
42 features	wt: 29 features, sl: 24 features, st: 3 features
84 features	wt: 57 features, sl: 48 features, st: 5 features
	Fully connected, 128-D
	Fully connected, 64-D
	Fully connected, 1-D

ing dataset (chosen deterministically). Xsertion built X-CNN topologies using KerasNet and FitNet4 as blueprints, using 80% of the given training set for unimodal training (for 80 epochs) and 20% of the training set as a validation set. Hyperparameters α of 1 and 2 and β of 2 and 4 were used for KerasNet and FitNet4, respectively. All experiments are repeated 5 times in order to report 95% confidence intervals.

For evaluating the X-LSTM, stratified 10-fold crossvalidation was performed on the full dataset (for the LSTM, X-LSTM as well as several further baselines):

- A classical *time-series forecasting technique* (ARIMA), to evaluate the benefits of performing machine learning in this task;
- Standard "*shallow*" approaches to machine learning: *support vector machines* (SVM) with the RBF kernel, *random forests* (RF) and *Gaussian hidden Markov models* (GHMM), to assess the benefits of using deep neural networks. The SVM has been augmented to produce probabilistic predictions by leveraging Platt scaling [132];
- Feedforward deep neural networks (DNN), expressing the relative necessity of using recurrent architectures;
- A recent state-of-the-art neural approach in processing multimodal sequential data [136] which imposes cross-modality through *weight sharing* of the recurrent weights ($\mathbf{U}_*$) across LSTM superlayers—I will refer to this method as *SH-LSTM*. This comes at a cost to expressivity—in order to share them, these weight matrices need to have

the same sizes, implying the different modality streams need to all compute the *same number of features* at each depth level.

All baselines' hyperparameters have been optimised using thorough sweeps. The non-sequential models (SVM, RF, DNN) were only trained on the last l days of the sequences, where $l = 10$ was found to give the best results. The SH-LSTM was optimised to have the same number of parameters as the baseline LSTM, and it was found that better results are achieved when sharing *between sleep and weight superlayers only*—as expected, given their superior unimodal predictive power.

ROC curves (and the associated *area* under them) are used as the primary evaluation metric, given the class imbalance of the dataset, and the fact that it is unclear how a practical classification threshold for this task should be chosen. For completeness, I also report the accuracy, precision, recall, F_1 score and the Matthews correlation coefficient [114] under the classification threshold which maximises the F_1 score. To confirm that the advantages demonstrated by the X-LSTM methodology are statistically significant, *paired t-testing* has been performed on the metrics of individual crossvalidation folds, choosing a significance threshold of $p < 0.05$.

3.3 Results

Having detailed the essential aspects of the experimental setups, I will now present the outcomes of the experiments, demonstrating outperformance of the proposed cross-modal architectures. These quantitative results have inspired several follow-up qualitative studies, the results of which will also be presented here.

3.3.1 Quantitative results

The results of the X-CNN quantitative studies[2] are presented in Tables 3.4–3.5. They demonstrate that X-CNN architectures consistently remain competitive against their baselines, and that these benefits are particularly pronounced when the datasets are increasingly *sparse* (strongest differences being observed on small CIFAR-100 training setups). Furthermore, the Xsertion library produced statistically significantly outperforming architectures in *all* of the settings considered—showing that, despite the rise in hyperparameter count for the early-fusion models, these hyperparameters can be pragmatically optimised in a seamless way, requiring little to no additional user input. As such, armed with a

[2] It should be noted that the results are as reported in the Xsertion paper by Laurynas Karazija [87]. The reason behind this is twofold: besides allowing for a clear comparison between hand-crafted X-CNNs and Xsertion, it also ameliorates the unstable results reported in the original X-CNN paper [176]—likely to be caused by an outdated version of the Theano deep learning framework used at the time.

Table 3.4: Comparison of accuracies of KerasNet based models, with highlighted 95% confidence intervals (after five independent runs).

CIFAR-10

Model $\backslash^{p\%}$	20% (%)	40% (%)	60% (%)	80% (%)	100% (%)
KerasNet	70.02 ± 0.14	76.57 ± 0.16	79.28 ± 0.17	81.40 ± 0.10	82.55 ± 0.11
X-KerasNet	71.00 ± 0.23	76.92 ± 0.10	79.62 ± 0.16	81.32 ± 0.10	82.68 ± 0.15
Xsertion	$\mathbf{72.02} \pm 0.70$	$\mathbf{77.29} \pm 0.16$	$\mathbf{79.92} \pm 0.07$	$\mathbf{81.54} \pm 0.10$	$\mathbf{82.93} \pm 0.09$

CIFAR-100

Model $\backslash^{p\%}$	20% (%)	40% (%)	60% (%)	80% (%)	100% (%)
KerasNet	28.20 ± 0.13	36.28 ± 0.24	42.14 ± 0.56	45.40 ± 0.33	48.53 ± 0.35
X-KerasNet	30.41 ± 0.32	39.32 ± 0.39	43.95 ± 0.40	47.08 ± 0.23	48.96 ± 0.22
Xsertion	$\mathbf{31.31} \pm 0.49$	$\mathbf{40.01} \pm 0.11$	$\mathbf{44.74} \pm 0.20$	$\mathbf{47.75} \pm 0.27$	$\mathbf{50.29} \pm 0.53$

baseline CNN and a dataset, any deep learning practitioner should be able to leverage Xsertion to obtain an outperforming X-CNN variant.

The ROC curves and related quantitative results for the X-LSTM quantitative study are summarised by Table 3.6 and Figure 3.5. Similarly as to the X-CNN, it was found that all of the observed differences in ROC-AUC are indeed statistically significant, verifying simultaneously that the recurrent models are superior to other baseline approaches, that the X-LSTM has significantly improved on its LSTM baselines. In spite of the various optimisations applied to the weight sharing strategy, SH-LSTM was unable to outperform the baseline LSTM—highlighting once again its lack of ability to accurately specify relative importances between modalities, which is essential for this task.

3.3.2 Qualitative results

Firstly, I will demonstrate that cross-connections inserted in the considered X-CNN models, though being 1×1 convolutions, learn more complex functions than simple feature map passing. First, we note that the weights of a 1×1 convolutional layer may be represented as a 2D table that maps input channels to output channels (akin to an adjacency matrix, where columns are the input channels and rows are the output channels).

Rather than displaying the raw table values, I decided to visualise weights in a *heatmap*

Table 3.5: Comparison of accuracies of FitNet4 based models, with highlighted 95% confidence intervals (after five independent runs).

CIFAR-10

Model$^{\backslash p\%}$	20% (%)	40% (%)	60% (%)	80% (%)	100% (%)
FitNet4	75.47 ± 0.32	82.02 ± 0.18	84.98 ± 0.20	86.22 ± 0.19	87.42 ± 0.05
X-FitNet4	76.56 ± 0.24	82.43 ± 0.07	85.11 ± 0.19	86.23 ± 0.18	87.42 ± 0.08
Xsertion	$\mathbf{77.35} \pm 0.15$	$\mathbf{82.66} \pm 0.09$	$\mathbf{85.43} \pm 0.12$	$\mathbf{86.78} \pm 0.16$	$\mathbf{87.77} \pm 0.22$

CIFAR-100

Model$^{\backslash p\%}$	20% (%)	40% (%)	60% (%)	80% (%)	100% (%)
FitNet4	29.29 ± 1.69	40.91 ± 2.48	50.94 ± 0.51	55.47 ± 0.96	58.92 ± 0.60
X-FitNet4	36.17 ± 0.27	48.02 ± 0.72	54.18 ± 0.36	57.98 ± 0.33	60.32 ± 0.29
Xsertion	$\mathbf{38.59} \pm 0.37$	$\mathbf{50.11} \pm 0.30$	$\mathbf{55.48} \pm 0.41$	$\mathbf{59.06} \pm 0.63$	$\mathbf{61.67} \pm 0.31$

style; Figure 3.6 showcases this visualisation for the first cross-connection layer of X-FitNet4 (trained on 100% of CIFAR-10's training set). Green colours indicate that an input channel has a positive connection weight to the respective output channel while blue colours indicate negative weights. The colour intensities are proportional to the absolute weight values.

It can be seen that each output channel of the cross-connection layer is obtained through a *nontrivial weighted combination* of input channels. It is therefore hypothesised that the cross-connection layers selectively filter and combine input features that are more utilisable in another processing stream.

To delve deeper into what kinds of features the cross-connection layers are filtering, combining and passing, the layer-wise feature-map activation techniques proposed by [159] were applied. This technique performs gradient ascent on a white-noise input image to maximise activations of a specific channel of feature maps at any of the layers within a pre-trained model. The objective function for gradient ascent is defined as

$$\mathbf{I}' = \underset{\mathbf{I}}{\operatorname{argmax}}\, \Sigma(\mathbf{I}) - \lambda \|\mathbf{I}\|^2 \tag{3.6}$$

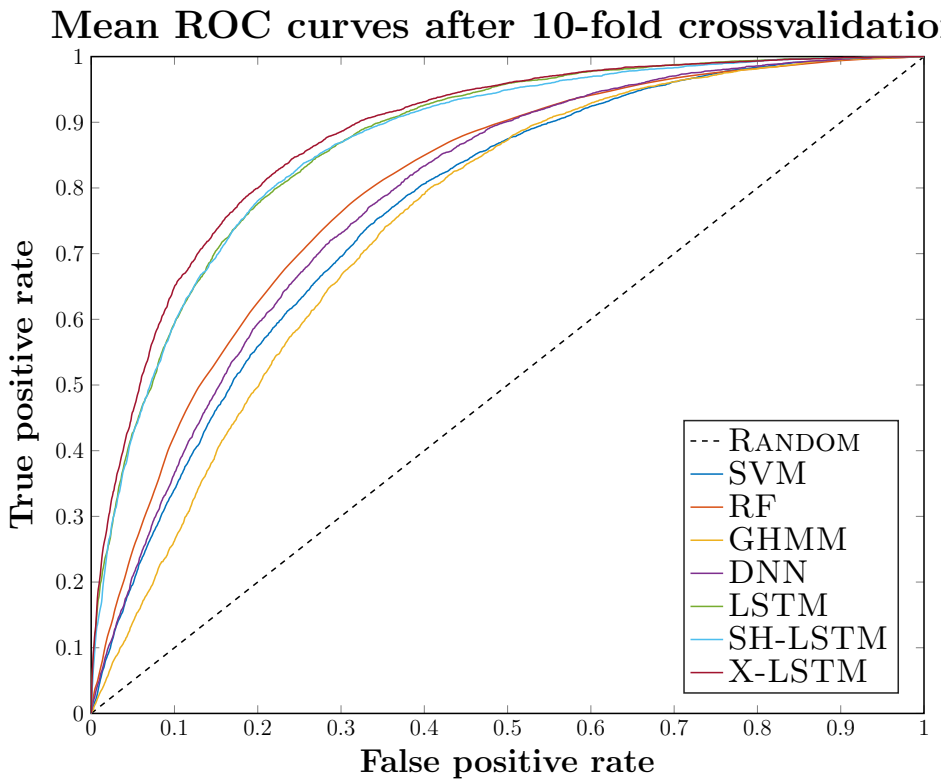


Figure 3.5: Mean ROC curves for the baselines, LSTM and the best-performing SH-LSTM and X-LSTM models.

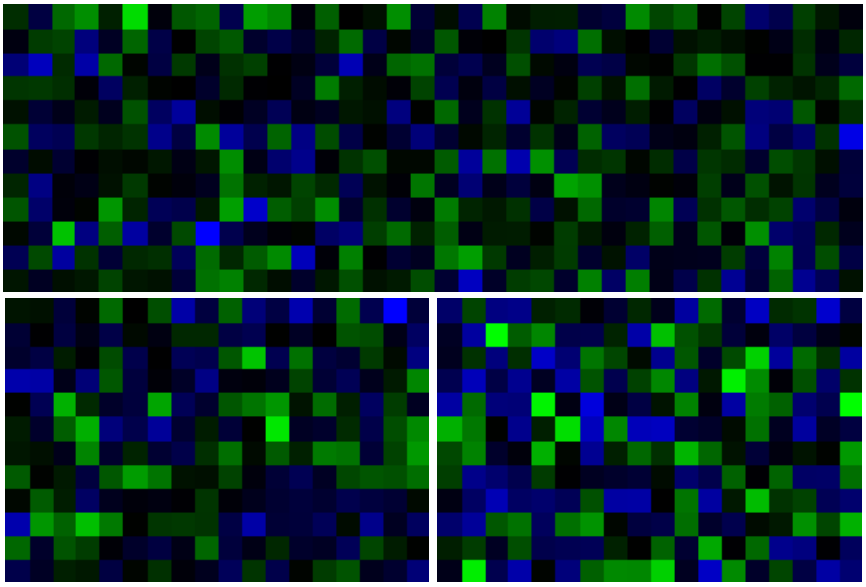


Figure 3.6: Weight visualisation of the first-level cross-connection layer for the X-FitNet4 CNN. The columns correspond to input channels, while rows correspond to output channels. Green colour indicates a positive-weight connection between an input channel and an output channel, while blue colour indicates a negative-weight connection. The colour intensities are proportional to the absolute weight values. **Top**: Y $\rightsquigarrow$ U/V (36 input channels, 12 output channels). **Bottom, left-to-right**: U $\rightsquigarrow$ Y and V $\rightsquigarrow$ Y (18 input channels, 12 output channels).

Table 3.6: Comparative evaluation results of the baseline models against the LSTMs after 10-fold crossvalidation. Reported p-values are for the X-LSTM vs. each baseline for the ROC-AUC metric.

Metric	ARIMA	SVM	RF	GHMM	DNN	LSTM	SH-LSTM	X-LSTM
Accuracy	59.22%	67.65%	70.97%	66.31%	68.93%	79.12%	78.49%	**80.30%**
Precision	42.96%	52.54%	56.05%	51.26%	53.80%	67.25%	65.31%	**68.66%**
Recall	50.44%	81.02%	81.34%	82.32%	**83.02%**	79.30%	82.95%	81.62%
F_1 score	46.40%	63.71%	66.25%	63.11%	65.18%	72.69%	72.98%	**74.37%**
MCC	13.94%	39.74%	44.75%	38.57%	42.63%	56.60%	56.80%	**59.45%**
ROC AUC	—	76.77%	79.97%	74.86%	78.54%	86.91%	86.63%	**88.07%**
p-value	—	$2 \cdot 10^{-12}$	$6 \cdot 10^{-10}$	$7 \cdot 10^{-11}$	$2 \cdot 10^{-11}$	$1 \cdot 10^{-4}$	$4 \cdot 10^{-5}$	—

where $\mathbf{I}$ is input image, $\Sigma(\mathbf{I})$ is the activation of the considered neuron when provided with $\mathbf{I}$ as input, and λ is a regularisation factor. After iterating for a number of gradient ascent steps, the original white-noise image will be modified into patterns that approximate the detection function of a specific neuron.

For the first cross-connection layer of a pre-trained X-FitNet4, I have visualised a selection of channel activations in Figure 3.7. This visualisation indicates that the cross-connection layer is indeed passing combined lower-level features, such as the addition of horizontal and vertical stripes in the upper right image in the figure. It can further be observed that the *pattern frequency* for the Y channel's cross-connection layer is higher than the one for the U and V layers. This observation reflects the fact that the human vision system is able to detect higher frequency variations in intensity than chrominance. This is a solid indicator that the X-CNN architecture, when faced with an image classification task in the YUV colour scheme, is actually attempting to *mimic human vision*.

For qualitatively analysing the learning potential of an X-LSTM, initially, note an obvious factor to the model's power: the *magnitude* of the weight objective the user has set. It should be expected that, at smaller weight objective magnitudes, the model's confusions will be roughly *evenly distributed* between successes and failures. As objectives become more ambitious, the model should become increasingly skewed towards *predicting failure more often* (as a consequence of the realistic statistics on such objectives), and therefore the majority of confusions are expected to represent successful sequences that the model misclassified as failures.

After aggregating the X-LSTM's model prediction across all crossvalidation folds (for

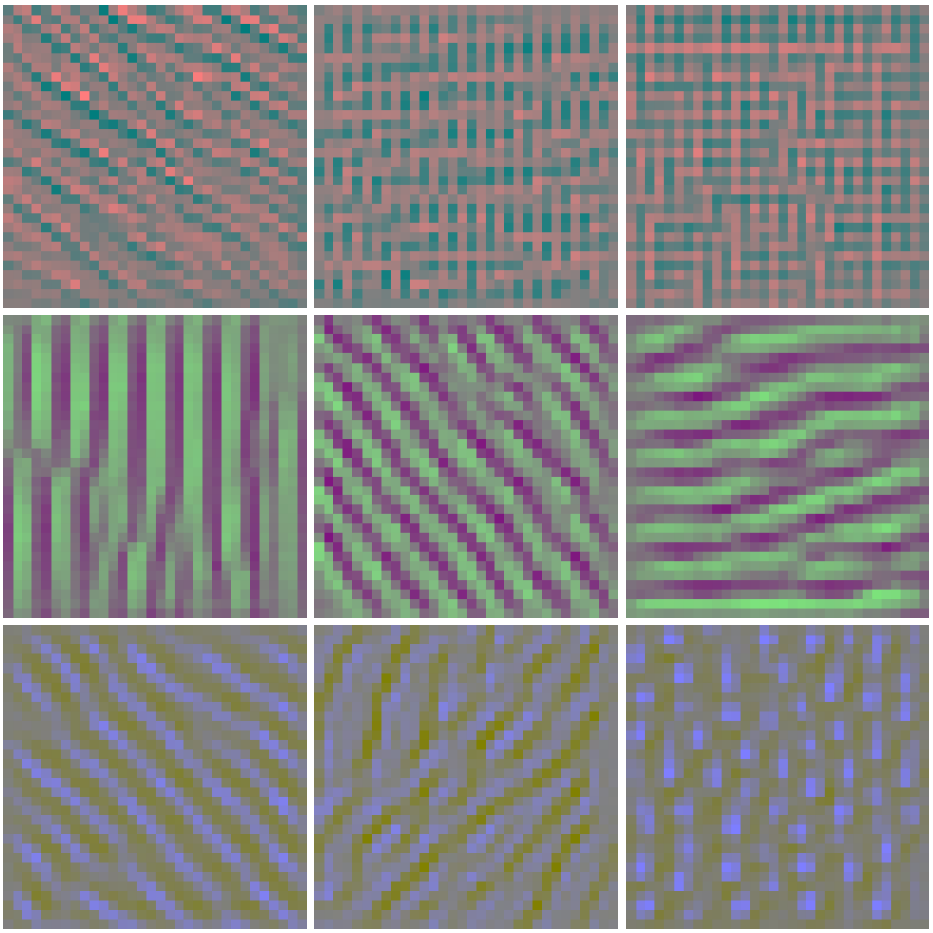


Figure 3.7: Artificially generated images (from white noise) that cause strong activations of specific channels in the first cross-connection layer of a pre-trained X-FitNet4 model. **Top**: Three channels from the Y $\rightsquigarrow$ U/V cross-connections. **Middle**: Three channels from the U $\rightsquigarrow$ Y cross-connections. **Bottom**: Three channels from the V $\rightsquigarrow$ Y cross-connections.

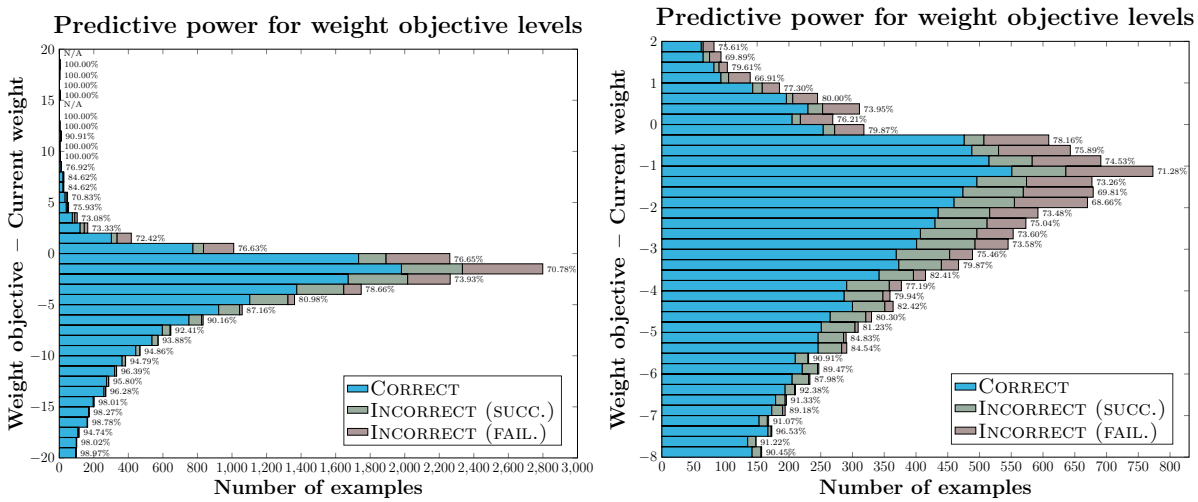


Figure 3.8: **Left:** A bar plot demonstrating the X-LSTM's performance for different magnitudes of weight objectives (at the classification threshold of 0.5). **Right:** The same plot, zoomed in on the $[-8\text{kg}, 2\text{kg}]$ range of weight objectives (where the majority of the examples are).

a classification threshold of 0.5), a histogram of the proportions of correctly classified, incorrectly classified successful and incorrectly classified failed sequences—for various bins of weight objective magnitudes—was produced (Figure 3.8). The results completely match the expectations outlined above—at lower weight objective magnitudes, there is a roughly 50/50 split between misclassified successes and failures. However, starting at -3kg and moving towards more ambitious objectives, there is a clear bias towards misclassifying successful sequences, which eventually grows into nearly all misclassified sequences being successful. This kind of behaviour is likely to be desirable—on average, it will encourage the users to choose *realistic* weight objectives, at the expense of making incorrect initial predictions about a few users that do eventually manage to achieve very ambitious goals.

Lastly, I apply the same approach used to visualise channel activations for X-CNN cross-connections [159] to visualise *classification models* for a pre-trained X-LSTM. Starting with a zero-sequence as input, gradient ascent was applied (following Equation 3.6) to produce a sequence that maximises or minimises the model's confidence in predicting success or failure.

The generated sequences spanning 10 days are shown in Figure 3.9. As expected, a user is deemed to be likely to hit their weight objective if there is a downwards trend in weight and an upwards trend in steps, and vice-versa for a failing sequence. Interestingly—and compounding the strong unimodal results for the sleep metrics—the model also uncovered that to have a higher confidence of success, it is important for the user to **fall asleep**

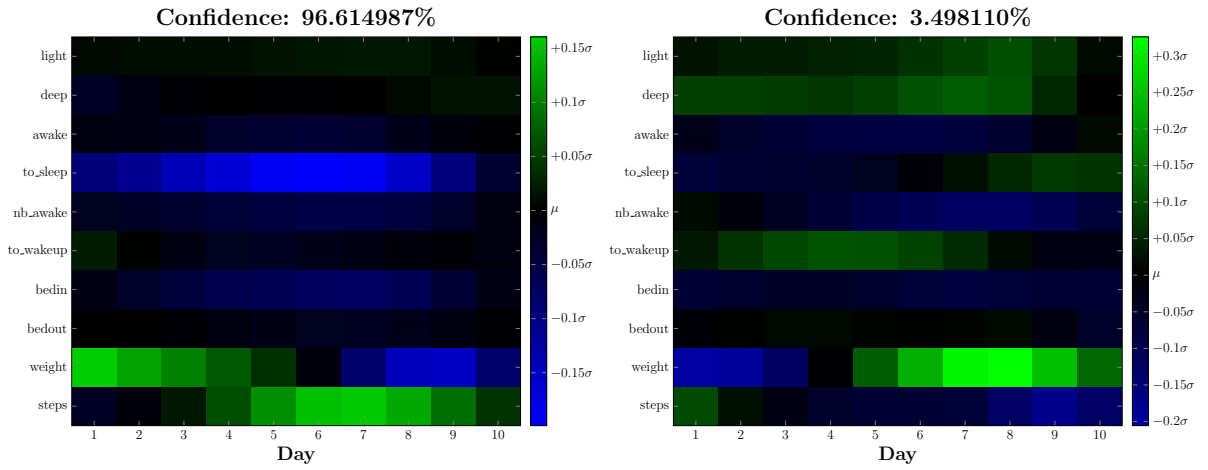


Figure 3.9: Iteratively produced artificial sequences that maximise the model's confidence in achieving (**left**) or failing (**right**) a -4kg weight objective.

quicker once going to bed. This is likely encoding important latent variables that can not directly be accessed from the dataset—for example, a person that takes more time to fall asleep is more likely to snack in the evening, which is known to be detrimental to weight loss (as previously observed in biomedical research [95, 122, 152]).

Aside from helping establish a more illustrated notion of the neural network's decision-making process, this analysis may also enhance feature selection. Namely, the features that are not significantly differently activated in the two cases may potentially be assumed to be *irrelevant*—here, those are likely to be the time spent awake during the night, number of times the person is awoken during the night, and the bed-in/bed-out times.

3.4 Generalisations: XFlow

Leading on from the successful results outlined in the preceding section, it may be natural to reflect on the initial challenge of early-fusion models that I have chosen to neglect at the beginning of this chapter: *is it tractable to design cross-connections for **arbitrary** input modality types*?

As a brief aside, at the end of this chapter I would like to highlight that the answer to this question is **affirmative**, as a result of the study performed by Cătălina Cangea (as her MPhil in Advanced Computer Science project performed under my co-supervision). The produced **XFlow** architecture [21] generalises cross-connections to *transcend* feature extractors of different dimensionalities—in this case, 1D and 2D signals in the context of *audiovisual classification*.

In this particular case, the visual input is given as *images* (to be processed by a CNN superlayer), and the audio input is given as *MFCC features* (to be processed by an MLP superlayer). Therefore, the function $\phi_\cdot$ of Equation 3.1 is a *convolutional layer* for the image modality and a *fully-connected layer* for the sound modality.

Designing a viable cross-connection in this case requires being able to reduce input of one dimensionality to the other. However, it turns out that these are already well-understood from other computer vision tasks:

- Going from a 2D representation to 1D occurs at the tail-end of most CNNs—as such, a suitable function $\chi_{img \rightsquigarrow snd}$ is a stack of *convolutional* and *pooling* layers, followed by a *flattening* operation.
- The converse operation occurs commonly in *encoder-decoder* architectures—such as the ones used for *image segmentation* tasks [4]. Therein, the input image is first reduced to a flat representation, and then a segmentation prediction is produced by reshaping the image from the flat representation. A suitable function $\chi_{snd \rightsquigarrow img}$ is, therefore, a *fully-connected layer* followed by a *reshaping* operation (to shape the output into 2D), followed by *deconvolution* (to perform the inverse operation to a convolutional layer).

It turns out that these intuitive definitions of cross-connections work well without substantial hyperparameter optimisation, and can be freely inserted into the network architecture (as illustrated by Figure 3.10), promptly leading to outperformance of baseline architectures (multimodal architectures lacking early fusion), as well as interpretable insights on the features being passed—such as the fact that the images synthesised by the 1D $\rightsquigarrow$ 2D cross-connection tend to mimic intermediate activations of a *lip-reading* network, while encoding changes in *volume* of the input audio track.

Such details are omitted from this dissertation in the interests of brevity, but illustrate that early fusion through cross-connection is a viable direction for future research, one that could feasibly compete with related recent approaches such as FiLM [130].

3.5 Summary

In this chapter, I have outlined my contributions towards allowing *early fusion* to be feasibly deployed within multimodal deep learning architectures. Restricting to the case where input modalities were all grid-like (yielding the **X-CNN** architecture) or sequential (yielding the **X-LSTM** architecture) allowed for a more robust and rigorous analysis of

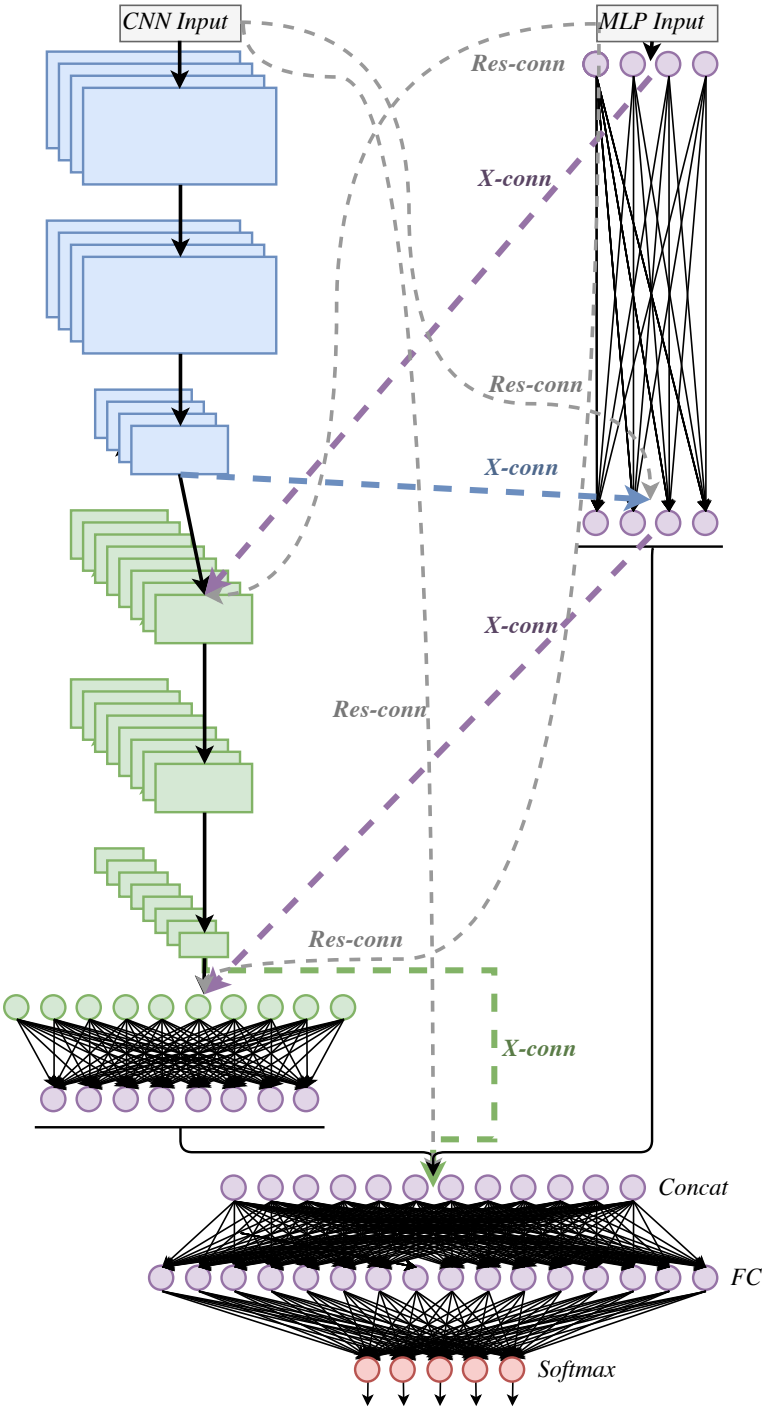


Figure 3.10: CNN × MLP XFlow model with cross-connections clearly highlighted. It has been shown that adding *residual cross-connections* (ones that directly shortcut the input of one modality to an intermediate representation of another) are highly beneficial to outperformance.

these benefits—and they have both been shown to significantly outperform their baseline approaches, especially under *challenging* data scenarios. The presented qualitative results further augment the insights of these findings, illustrating that interpretable behaviour can be extracted from these models.

Specially, subsequent work performed by Laurynas Karazija and Cătălina Cangea under my co-supervision has demonstrated that the primary issues of such approaches (the quadratic increase in hyperparameter space and difficulty of generalising to modalities of different dimensionalities) can both be ameliorated—through their contributions to developing the **Xsertion** library and the **XFlow** architecture.

Chapter 4

Attentive graph convolutions

> "The image of the world around us, which we carry in our head, is just a *model*. Nobody in his head imagines all the world, government or country. He has only selected *concepts*, and *relationships* between them, and uses those to represent the real system."
>
> Jay Wright Forrester

A multitude of important real-world datasets come together with some form of graph structure: *social networks*, *citation networks*, *protein-protein interactions*, *brain connectome data*, etc. Extending neural networks to be able to properly deal with this kind of data is therefore a very important direction for machine learning research [6, 16, 65], but one that has received comparatively rather low levels of attention until very recently. In the remainder of my dissertation, I will dedicate undivided attention to my own contributions towards this challenge.

In this chapter, I will introduce *Graph Attention Networks* (GATs) [174], a graph convolutional layer leveraging *masked self-attentional layers* [173] to address the theoretical shortcomings of prior methods based on graph convolutions and their approximations. To the best of my knowledge, at the time of publication this was the first proposed and evaluated graph convolutional layer to satisfy all of the desirable properties for a generic convolution layer simultaneously. These theoretical features are validated on several benchmarks, demonstrating results competitive with—and often exceeding—the state-of-the-art at the time of publication. I will also outline the results of this validation study.

Finally, I will outline several pieces of relevant work that leverage or further build on

GAT(-like) models—a list which is by no means exhaustive, but aims to demonstrate the real-world utility of the model. It should be noted, as well, that attentive models on graphs have recently exclusively been covered in a separate *survey paper* [105].

4.1 Prior approaches

Convolutional Neural Networks (CNNs) have been successfully applied to tackle problems such as image classification [68], semantic segmentation [80] or machine translation [48], where the underlying data representation has a grid-like structure. These architectures efficiently reuse their local filters, with learnable parameters, by applying them to all the input positions.

However, many interesting tasks involve data that can not be represented in a grid-like structure and that instead lies in an irregular domain. This is the case of 3D meshes, social networks, telecommunication networks, biological networks or brain connectomes. Such data can usually be represented in the form of graphs.

There have been several attempts in the literature to extend neural networks to deal with arbitrarily structured graphs. Early work used recursive neural networks to process data represented in graph domains as directed acyclic graphs [44, 161]. Graph Neural Networks (GNNs) were introduced in [58] and [154] as a generalisation of recursive neural networks that can directly deal with a more general class of graphs, e.g. cyclic, directed and undirected graphs. GNNs consist of an iterative process, which propagates the node states until equilibrium; followed by a neural network, which produces an output for each node based on its state. This idea was adopted and improved by [107], which proposes to use gated recurrent units [24] in the propagation step.

Nevertheless, there is an increasing interest in generalising convolutions to the graph domain. Advances in this direction can often be categorised into two groups: *spectral* approaches and *spatial* approaches.

On one hand, spectral approaches work with a spectral representation of the graphs and have been successfully applied in the context of node classification. In [17], the convolution operation is defined in the Fourier domain by computing the eigendecomposition of the graph Laplacian, resulting in potentially intense computations and non-spatially localised filters. These issues were addressed by subsequent works. [69] introduced a parameterisation of the spectral filters with smooth coefficients in order to make them spatially localised. Later, [33] proposed to approximate the filters by means of a Cheby-

shev expansion of the graph Laplacian, removing the need to compute the eigenvectors of the Laplacian and yielding spatially localised filters. Finally, [92] simplified the previous method by restricting the filters to operate in a 1-step neighbourhood around each node. However, in all of the aforementioned spectral approaches, the learned filters depend on the Laplacian eigenbasis, which depends on the graph structure. Thus, a model trained on a specific structure can not be directly applied to a graph with a different structure.

On the other hand, we have spatial approaches [1, 39, 64], which define convolutions directly on the graph, operating on groups of spatially close neighbours. One of the challenges of these approaches is to define an operator which works with different sized neighbourhoods and maintains the weight sharing property of CNNs. In some cases, this requires learning a specific weight matrix for each node degree [39], using the powers of a transition matrix to define the neighbourhood while learning weights for each input channel and neighbourhood degree [1], or extracting and normalising neighbourhoods containing a fixed number of nodes [125]. [121] presented mixture model CNNs (MoNet), a spatial approach which provides a unified generalisation of CNN architectures to graphs. More recently, [64] introduced GraphSAGE, a method for computing node representations in an *inductive* manner. This technique operates by sampling a fixed-size neighbourhood of each node, and then performing a specific aggregator over it (such as the mean over all the sampled neighbours' feature vectors, or the result of feeding them through a recurrent neural network). This approach has yielded impressive performance across several large-scale inductive benchmarks.

4.2 What's in a *graph* convolution?

CNNs are a major workforce when it comes to working with image data. They exploit the fact that images have a highly rigid and regular connectivity pattern (each pixel "connected" to its eight neighbouring pixels), making such an operator *trivial* to deploy (as a small kernel matrix which is slid across the image).

Arbitrary graphs are a **much harder** challenge! Ideally, we would like to aggregate information across each of the nodes' neighbourhoods in a principled manner, but we are no longer guaranteed such rigidity of structure.

Reflecting on the desirable traits of image convolutions, we arrive at the following properties we would ideally like a graph convolutional layer to have:

- **Computational and storage efficiency** (requiring no more than $O(V + E)$ time and memory, where V is the number of nodes and E the number of edges);

- **Fixed** number of parameters (independent of input graph size);
- **Localisation** (acting on a *local neighbourhood* of a node);
- Ability to specify **arbitrary importances** to different neighbours;
- Applicability to **inductive problems** (arbitrary, unseen graph structures).

Satisfying all of the above at once has proved to be quite challenging, and indeed, none of the prior techniques outlined above have successfully achieved them simultaneously:

- With respect to spectral approaches, the original Graph Fourier Transform approach of [17] suffers from lack of localisation, computational inefficiencies, and the fact that the number of parameters is not fixed.
- Subsequent simplifications of this framework [33, 92] address all of these issues, at the expense of becoming unable to specify arbitrary importances to different neighbours (these importances being fully conditioned on the graph structure).
- In fact, *any* spectral method (one dependent on knowledge of the graph Laplacian) is unlikely to be theoretically applicable to inductive problems—given that their learnt filters are dependent on the Laplacian and may not be applicable to *arbitrary* graph structures. Furthermore, the Laplacian may not be known upfront for many graphs of interest, especially *dynamic* graphs.
- The early spatial approach of [107] uses a *recurrent* update rule, which not only seems unnatural for *spatially* defined data, but also requires the number of features computed at every stage to be *fixed*—therefore restricting the neural network practitioner from specifying a true analogue of a convolutional layer (with arbitrary numbers of filters).
- The spatial approach of [39] handles the issue of varying degrees in a graph by specifying a *separate* convolutional filter for each individual node degree. While this was suitable in their particular application of interest (fingerprinting of *small chemicals*, wherein no atom may be connected to more than *five* other atoms), it fails to generalise to graphs with wide degree distributions.
- Lastly, GraphSAGE [64] defines several spatial graph convolutional layers with *inductive* properties. This includes:
  - Several reparametrisations of the GCN update rule, which still suffer from being unable to specify different importances to different neighbours;
  - An LSTM-based aggregator, which is not easy to parallelise and lacks *permutation invariance* over the neighbourhood;

– A *max-pool* based aggregator, which relies on being able to subsample neighbourhoods (as applying it in parallel across the full neighbourhood of a node is nontrivial to perform sparsely). This has the limitation of requiring subsampling at training as well as *inference* time—thus restricting the amount of information available to the model.

It should be noted that there existed two proposed spatial frameworks—the MoNet [121] and MPNNs [49]—that can generalise many other spatial approaches. Therefore—in their general form—they do satisfy all the desirable properties of graph convolutions. However, for both of them, the experimentally tested architectures always reduced to one of the previously proposed ones—thus sharing their theoretical limitations.

4.3 The self-attentional solution

In my opinion, the common issue of most previously considered methods is that they defined the recombination weights across their neighbourhood *explicitly*[1] (either based on the structural properties of the graph or as a learnable weight), which required compromising at least one other desirable property. Thus, I believe that the optimal graph convolutional layers will always specify these weights *implicitly*—and in this chapter I will propose **self-attention** as a *scalable* way of doing so.

Attention mechanisms have become almost a *de facto* standard in many sequence-based tasks [5, 48]. One of the benefits of attention mechanisms is that they allow for dealing with variable sized inputs, focussing on the most relevant parts of the input to make decisions. When an attention mechanism is used to compute a representation of a single sequence, it is commonly referred to as *self-attention* or *intra-attention*. Together with Recurrent Neural Networks (RNNs) or convolutions, self-attention has proven to be useful for tasks such as machine reading [23] and learning sentence representations [109]. However, [173] showed that not only *self-attention* can improve a method based on RNNs or convolutions, but also that it is sufficient for constructing a powerful model obtaining *state-of-the-art* performance on the machine translation task.

Inspired by this recent work, I introduce an attention-based architecture to perform node classification of graph-structured data. The idea is to compute the hidden representations of each node in the graph, by attending over its neighbours, following a *self-attention*

[1] A notable exception is the MoNet framework [121] which our work can be considered an instance of. However, in all of the experiments presented in this paper, the weights were implicitly specified based on the graph's structural properties, meaning that two nodes with same local topologies would always necessarily receive the same (learned) weighting—thus still violating one of the desirable properties for a graph convolution.

strategy. The attention architecture has several interesting properties: (1) the operation is efficient, since it is parallelisable across node-neighbour pairs; (2) it can be applied to graph nodes having different degrees by specifying arbitrary weights to the neighbours; and (3) the model is directly applicable to *inductive* learning problems, including tasks where the model has to generalise to completely unseen graphs.

It is worth noting that, as is the case for [1] and [92], this work can also be reformulated as a particular instance of MoNet [121]. Moreover, the approach of sharing a neural network computation across edges is reminiscent of the formulation of relational networks [151] and VAIN [75], wherein relations between objects or agents are aggregated pairwise, by employing a shared mechanism. Similarly, our proposed attention model can be connected to the works by [34] and [37], which use a neighbourhood attention operation to compute attention coefficients between different objects in an environment. Other related approaches include locally linear embedding (LLE) [142] and memory networks [186]. LLE selects a fixed number of neighbours around each data point, and learns a weight coefficient for each neighbour to reconstruct each point as a weighted sum of its neighbours. A second optimisation step extracts the point's feature embedding. Memory networks can be related to this work, in particular, if the neighbourhood of a node is interpreted as the memory, which is used to compute the node features by attending over its values, and then is updated by storing the new features in the same position.

4.4 GAT architecture

In this section, I will present the building block layer used to construct arbitrary graph attention networks, and directly outline its theoretical and practical benefits and limitations compared to prior work in the domain of neural graph processing.

4.4.1 Graph attentional layer

I will start by describing a single *graph attentional layer*, as the sole layer utilised throughout all of the GAT architectures used in the experimental evaluation. The particular attentional setup utilised closely follows the work of [5]—but the framework is agnostic to the particular choice of attention mechanism.

The input to the layer is a set of node features, $\mathbf{H} = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}, \vec{h}_i \in \mathbb{R}^F$, where N is the number of nodes, and F is the number of features in each node. The layer produces a new set of node features (of potentially different cardinality F'), $\mathbf{H}' = \{\vec{h}'_1, \vec{h}'_2, \ldots, \vec{h}'_N\}, \vec{h}'_i \in \mathbb{R}^{F'}$, as its output.

In order to obtain sufficient expressive power to transform the input features into higher-level features, at least one learnable linear transformation is required. To that end, as an initial step, a shared linear transformation, parametrised by a *weight matrix*, $\mathbf{W} \in \mathbb{R}^{F' \times F}$, is applied to every node. Then, *self-attention* is performed on the nodes—a shared attentional mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$ computes *attention coefficients*

$$e_{ij} = a(\vec{h}_i, \vec{h}_j) \tag{4.1}$$

that indicate the *importance* of node j's features to node i. In its most general formulation, the model allows every node to attend on every other node, *dropping all structural information*. The graph structure is injected into the mechanism by performing *masked attention*—we only compute e_{ij} for nodes $j \in \mathcal{N}_i$, where $\mathcal{N}_i$ is some *neighbourhood* of node i in the graph. In all experiments, these will be exactly the first-order neighbours of i (including i). To make coefficients easily comparable across different nodes, they are normalised across all choices of j using the softmax function:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})} \tag{4.2}$$

For all experiments, the attention mechanism a is a single-layer feedforward neural network, parametrised by a weight vector $\vec{a} \in \mathbb{R}^{2F'}$, and applying the LeakyReLU nonlinearity (with negative input slope $\alpha = 0.2$). Fully expanded out, the coefficients computed by the attention mechanism (illustrated by Figure 4.1 (left)) may then be expressed as:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{a}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(\vec{a}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)} \tag{4.3}$$

where $\cdot^T$ represents transposition and $\|$ is the concatenation operation.

Once obtained, the normalised attention coefficients are used to compute a linear combination of the features corresponding to them, to serve as the final output features for every node (after potentially applying a nonlinearity, σ):

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W}\vec{h}_j\right) \tag{4.4}$$

To stabilise the learning process of self-attention, it has been found that extending the mechanism to employ *multi-head attention* is beneficial, similarly to [173]. Specifically, K independent attention mechanisms execute the transformation of Equation 4.4, and then

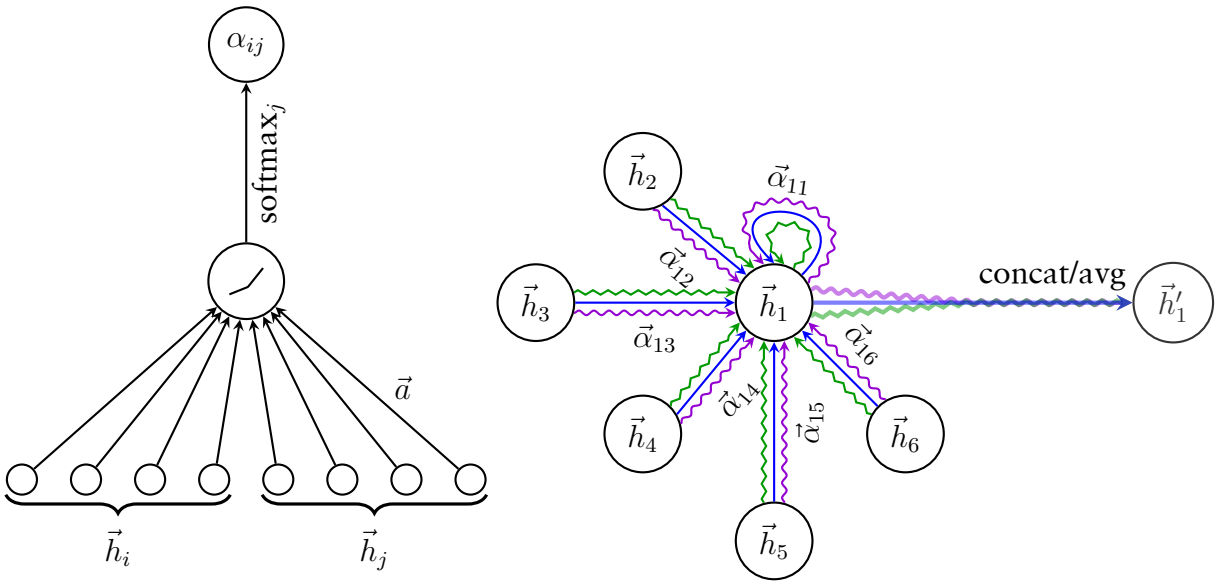


Figure 4.1: **Left**: The attention mechanism $a(\vec{h}_i, \vec{h}_j)$ employed by the GAT model, parametrized by a weight vector $\vec{a} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation. **Right**: An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$.

their features are concatenated, resulting in the following output feature representation:

$$\vec{h}'_i = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^{(k)} \mathbf{W}^{(k)} \vec{h}_j\right) \tag{4.5}$$

where $\|$ represents concatenation, $\alpha_{ij}^{(k)}$ are normalized attention coefficients computed by the k-th attention mechanism ($a^{(k)}$), and $\mathbf{W}^{(k)}$ is the corresponding input linear transformation's weight matrix. Note that, in this setting, the final returned output, $\mathbf{H}'$, will consist of KF' features (rather than F') for each node.

Specially, if we perform multi-head attention on the final (prediction) layer of the network, concatenation is no longer sensible—instead, *averaging* is employed, with the application of the final nonlinearity (usually a softmax or logistic sigmoid for classification problems) delayed until then:

$$\vec{h}'_i = \sigma\left(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in \mathcal{N}_i} \alpha_{ij}^{(k)} \mathbf{W}^{(k)} \vec{h}_j\right) \tag{4.6}$$

The aggregation of a multi-head graph attentional layer is illustrated by Figure 4.1 (right).

4.4.2 Theoretical properties

The graph attentional layer described in subsection 4.4.1 directly addresses several issues that were present in prior approaches to modelling graph-structured data with neural networks. In this section, I will enumerate the favourable properties of the GAT layer in turn, along with a discussion of a few of its limitations.

- Computationally, the GAT layer is highly **efficient**: the operation of the self-attentional layer can be parallelised across all edges, and the computation of output features can be parallelised across all nodes. No eigendecompositions or similar computationally intensive matrix operations are required. The time complexity of a single GAT attention head computing F' features may be expressed as $O(VFF' + EF')$, where F is the number of input features, and V and E are the numbers of nodes and edges in the graph, respectively. This complexity is on par with the baseline methods such as Graph Convolutional Networks (GCNs) [92].

- Furthermore, the GAT layer can be expressed solely in *sparse* matrix operations, reducing the storage complexity to linear in the number of nodes and edges and enabling the execution of GAT models on larger graph datasets.

- Applying multi-head attention multiplies the storage and parameter requirements by a factor of K, while the individual heads' computations are fully independent and can be parallelised.

- The GAT layer boasts a *fixed* number of parameters (all contained within the attentional mechanism and pointwise transformation), and it is trivially *localised* (as we only allow attending over neighbourhoods).

- As opposed to many prior approaches, GAT allows for (implicitly) assigning *different importances* to nodes of a same neighbourhood, enabling a leap in model capacity. Furthermore, analysing the learned attentional weights may lead to benefits in interpretability, as was the case in the machine translation domain (e.g. the qualitative analysis of [5]).

- The attention mechanism is applied in a shared manner to all edges in the graph, and therefore it does not depend on upfront access to the global graph structure or (features of) all of its nodes (a limitation of many prior techniques). This makes GATs directly applicable to **inductive** learning—including tasks where the model is evaluated on graphs that are *completely unseen* during training.

- GraphSAGE [64] samples a *fixed-size neighbourhood* of each node, in order to keep its computational footprint consistent; this does not allow it access to the entirety

of the neighbourhood while performing inference. Conversely, GAT works with the entirety of the neighbourhood (at the expense of a variable computational footprint, which is still on-par with methods like the GCN).

- As mentioned in Section 4.3, GAT can be reformulated as a particular instance of MoNet [121]. Specifically, setting the pseudo-coordinate function to be $u(x, y) = f(x) \| f(y)$, where $f(x)$ represent (MLP-transformed) features of node x and $\|$ is concatenation; and the weight function to be $w_j(u) = \text{softmax}(\text{MLP}(u))$ (with the softmax performed over the entire neighborhood of a node) would make MoNet's patch operator similar to the GAT layer. Nevertheless, one should note that, in comparison to previously considered MoNet instances, the GAT layer uses node features for similarity computations, rather than the node's structural properties (which would assume knowing the graph structure upfront).

Depending on the regularity of the graph structure in place, GPUs may not be able to offer major performance benefits compared to CPUs in the sparse scenarios considered here. It should also be noted that the size of the "receptive field" of graph attention networks is upper-bounded by the depth of the network (similarly as for GCN and similar models). Techniques such as skip connections [68] could be readily applied for appropriately extending the depth, however. Lastly, parallelisation across all the graph edges, especially in a distributed manner, may involve a lot of redundant computation, as the neighbourhoods will often highly overlap in graphs of interest.

4.5 Evaluation

I have performed comparative evaluation of GAT models against a wide variety of strong baselines and previous approaches, on four established graph-based benchmark tasks (transductive as well as inductive), achieving or matching state-of-the-art performance across all of them. This section summarises the experimental setup, results, and a brief qualitative analysis of a GAT model's extracted feature representations.

4.5.1 Datasets

Transductive learning I utilise three standard citation network benchmark datasets—Cora, Citeseer and Pubmed [156]—and closely follow the transductive experimental setup of [190]. In all of these datasets, nodes correspond to documents and edges to (undirected) citations. Node features correspond to elements of a bag-of-words representation of a document. Each node has a class label. Only 20 nodes per class are allowed to be used for training—however, honouring the transductive setup, the training algorithm

Table 4.1: Summary of the datasets used in the experiments.

	Cora	Citeseer	Pubmed	PPI
Task	Transductive	Transductive	Transductive	Inductive
# Nodes	2,708	3,327	19,717	56,944 (24 graphs)
# Edges	5,429	4,732	44,338	818,716
# Fts./Node	1,433	3,703	500	50
# Classes	7	6	3	121 (multilabel)
# Train. Nodes	140	120	60	44,906 (20 graphs)
# Val. Nodes	500	500	500	6,514 (2 graphs)
# Test Nodes	1,000	1,000	1,000	5,524 (2 graphs)

has access to all of the nodes' feature vectors. The predictive power of the trained models is evaluated on 1,000 test nodes, and 500 additional nodes are used for validation purposes (the same ones as used by [92]). The Cora dataset contains 2,708 nodes, 5,429 edges, 7 classes and 1,433 features per node. The Citeseer dataset contains 3,327 nodes, 4,732 edges, 6 classes and 3,703 features per node. The Pubmed dataset contains 19,717 nodes, 44,338 edges, 3 classes and 500 features per node.

Inductive learning I make use of a protein-protein interaction (PPI) dataset that consists of graphs corresponding to different human tissues [200]. The dataset contains 20 graphs for training, 2 for validation and 2 for testing. Critically, testing graphs remain *completely unobserved* during training. To construct the graphs, the preprocessed data provided by [64] was used. The average number of nodes per graph is 2,372. Each node has 50 features that are composed of positional gene sets, motif gene sets and immunological signatures. There are 121 potential labels for each node set from gene ontology, collected from the Molecular Signatures Database [165], and a node can possess several labels simultaneously.

An overview of the interesting characteristics of the datasets is given in Table 5.1.

4.5.2 Baseline methods

Transductive learning For transductive learning tasks, I compare against the same strong baselines and state-of-the-art approaches as specified in [92]. This includes label propa-

gation (LP) [198], semi-supervised embedding (SemiEmb) [187], manifold regularization (ManiReg) [10], skip-gram based graph embeddings (DeepWalk) [131], the iterative classification algorithm (ICA) [110] and Planetoid [190]. I also directly compare the GAT model model against GCNs [92], as well as graph convolutional models utilising higher-order Chebyshev filters [33], and the MoNet model presented in [121].

Inductive learning For the inductive learning task, I compare against the four different supervised GraphSAGE inductive methods presented in [64]. These provide a variety of approaches to aggregating features within a sampled neighbourhood: GraphSAGE-GCN (which extends a graph convolution-style operation to the inductive setting), GraphSAGE-mean (taking the elementwise mean value of feature vectors), GraphSAGE-LSTM (aggregating by feeding the neighbourhood features into an LSTM) and GraphSAGE-pool (taking the elementwise maximisation operation of feature vectors transformed by a shared nonlinear multilayer perceptron). The other transductive approaches are either completely inappropriate in an inductive setting or assume that nodes are incrementally added to a single graph, making them unusable for the setup where test graphs are completely unseen during training (such as the PPI dataset).

Additionally, for both tasks I provide the performance of a per-node shared multilayer perceptron (MLP) classifier (that does not incorporate graph structure at all).

4.5.3 Experimental setup

Transductive learning For the transductive learning tasks, I apply a two-layer GAT model. Its architectural hyperparameters have been optimised on the Cora dataset and are then reused for Citeseer. The first layer consists of $K = 8$ attention heads computing $F' = 8$ features each (for a total of 64 features), followed by an exponential linear unit (ELU) [26] nonlinearity. The second layer is used for classification: a single attention head that computes C features (where C is the number of classes), followed by a softmax activation. For coping with the small training set sizes, regularisation is liberally applied within the model. During training, L_2 regularisation with $\lambda = 0.0005$ is applied. Furthermore, dropout [162] with $p = 0.6$ is applied to both layers' inputs, as well as *to the normalised attention coefficients* (critically, this means that at each training iteration, each node is exposed to a stochastically sampled neighbourhood). Similarly as observed by [121], I found that Pubmed's training set size (60 examples) required slight changes to the GAT architecture: I have applied $K = 8$ output attention heads (instead of one), and strengthened the L_2 regularisation to $\lambda = 0.001$. Otherwise, the architecture matches the one used for Cora and Citeseer.

Inductive learning For the inductive learning task, I apply a three-layer GAT model. Both of the first two layers consist of $K = 4$ attention heads computing $F' = 256$ features (for a total of 1024 features), followed by an ELU nonlinearity. The final layer is used for (multi-label) classification: $K = 6$ attention heads computing 121 features each, that are averaged and followed by a logistic sigmoid activation. The training sets for this task are sufficiently large and I found no need to apply L_2 regularisation or dropout—I have, however, successfully employed skip connections [68] across the intermediate attentional layer. I utilise a batch size of 2 graphs during training. To strictly evaluate the benefits of applying an attention mechanism in this setting (i.e. comparing with a near GCN-equivalent model), I also provide the results when a *constant attention mechanism*, $a(x, y) = 1$, is used, with the same architecture—this will assign the same aggregation weight to every neighbour.

Both models are initialised using Xavier initialisation [50] and trained to minimise cross-entropy on the training nodes using the Adam SGD optimiser [88] with an initial learning rate of 0.01 for Pubmed, and 0.005 for all other datasets. In both cases, I use an early stopping strategy on both the cross-entropy loss and accuracy (transductive) or micro-F_1 (inductive) score on the validation nodes, with a patience of 100 epochs.

4.5.4 Results

The results of the comparative evaluation experiments are summarised in Tables 4.2–4.3.

For the transductive tasks, I report the mean classification accuracy (with standard deviation) on the test nodes of the GAT model after 100 runs, and reuse the metrics already reported in [92] and [121] for state-of-the-art techniques. Specifically, for the Chebyshev filter-based approach [33], I provide the maximum reported performance for filters of orders $K = 2$ and $K = 3$. In order to fairly assess the benefits of the attention mechanism, I further evaluate a GCN model that computes 64 hidden features, attempting both the ReLU and ELU activation, and reporting (as GCN-64*) the better result after 100 runs (which was the ReLU in all three cases).

For the inductive task, I report the micro-averaged F_1 score on the nodes of the two unseen test graphs, averaged after 10 runs, and reuse the metrics already reported in [64] for the other techniques. Specifically, as the learning setup is supervised, I compare against the supervised GraphSAGE approaches. To evaluate the benefits of aggregating across the entire neighbourhood, I further provide (as GraphSAGE*) the best result I was able to achieve with GraphSAGE by just modifying its architecture (this was with a three-layer GraphSAGE-LSTM with [512, 512, 726] features computed in each layer and 128

Table 4.2: Summary of results in terms of classification accuracies, for Cora, Citeseer and Pubmed. GCN-64* corresponds to the best GCN result computing 64 hidden features (using ReLU or ELU).

	Transductive		
Method	**Cora**	**Citeseer**	**Pubmed**
MLP	55.1%	46.5%	71.4%
ManiReg [10]	59.5%	60.1%	70.7%
SemiEmb [187]	59.0%	59.6%	71.7%
LP [198]	68.0%	45.3%	63.0%
DeepWalk [131]	67.2%	43.2%	65.3%
ICA [110]	75.1%	69.1%	73.9%
Planetoid [190]	75.7%	64.7%	77.2%
Chebyshev [33]	81.2%	69.8%	74.4%
GCN [92]	81.5%	70.3%	**79.0%**
MoNet [121]	81.7 ± 0.5%	—	78.8 ± 0.3%
GCN-64*	81.4 ± 0.5%	70.9 ± 0.5%	**79.0** ± 0.3%
GAT (ours)	**83.0** ± 0.7%	**72.5** ± 0.7%	**79.0** ± 0.3%

Table 4.3: Summary of results in terms of micro-averaged F_1 scores, for the PPI dataset. GraphSAGE* corresponds to the best GraphSAGE result I was able to obtain by just modifying its architecture. Const-GAT corresponds to a model with the same architecture as GAT, but with a constant attention mechanism (assigning same importance to each neighbour; GCN-like inductive operator).

	Inductive
Method	**PPI**
Random	0.396
MLP	0.422
GraphSAGE-GCN [64]	0.500
GraphSAGE-mean [64]	0.598
GraphSAGE-LSTM [64]	0.612
GraphSAGE-pool [64]	0.600
GraphSAGE*	0.768
Const-GAT (ours)	0.934 ± 0.006
GAT (ours)	$\mathbf{0.973} \pm 0.002$

features used for aggregating neighbourhoods). Finally, I report the 10-run result of the constant attention GAT model (as Const-GAT), to fairly evaluate the benefits of the attention mechanism against a GCN-like aggregation scheme (with the same architecture).

The reported results successfully demonstrate state-of-the-art performance being achieved or matched across all four datasets—in concordance with expectations, as per the discussion in Section 4.4.2. More specifically, the GAT model improved upon GCNs by a margin of 1.5% and 1.6% on Cora and Citeseer, respectively, suggesting that assigning different weights to nodes of a same neighbourhood may be beneficial. It is worth noting the improvements achieved on the PPI dataset: the GAT model improves by 20.5% w.r.t. the best GraphSAGE result I was able to obtain, demonstrating that the model has the potential to be applied in inductive settings, and that larger predictive power can be leveraged by observing the entire neighbourhood. Furthermore, it improves by 3.9% w.r.t. Const-GAT (the identical architecture with constant attention mechanism), once again directly demonstrating the significance of being able to assign different weights to different neighbours.

The effectiveness of the learned feature representations may also be investigated qualitatively, and for this purpose I provide a visualisation of the t-SNE [113]-transformed feature representations extracted by the first layer of a GAT model pre-trained on the Cora dataset (Figure 4.2). The representation exhibits discernible clustering in the projected 2D space. Note that these clusters correspond to the seven labels of the dataset, verifying the model's discriminative power across the seven topic classes of Cora. Additionally, I visualise the relative strengths of the normalised attention coefficients (averaged across all eight attention heads).

4.6 Extensions and applications

Following the publication of the original GAT paper [174], I have been delighted to witness (and contribute to) several new lines of research wherein GAT-like architectures have been leveraged to solve challenging problems. In this section, I will outline two subsequent contributions that I have personally been involved in, and outline several interesting contributions by others.

4.6.1 Mesh-based parcellation of the cerebral cortex

In this work—primarily investigated by Guillem Cucurull—we have considered the task of *cortical mesh segmentation* (predicting functional regions for locations on a human

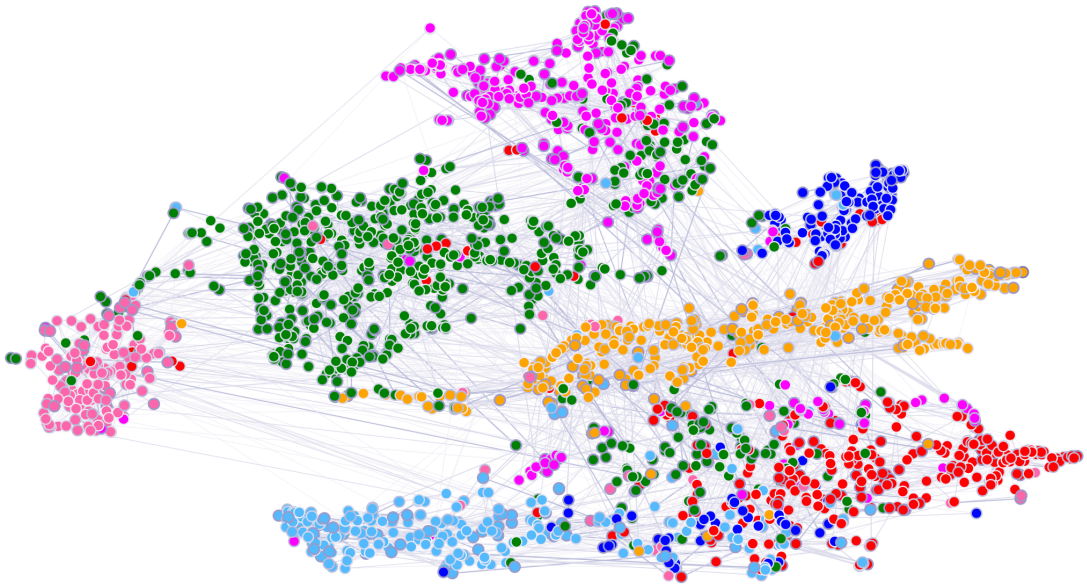


Figure 4.2: A t-SNE plot of the computed feature representations of a pre-trained GAT model's first hidden layer on the Cora dataset. Node colours denote classes. Edge thickness indicates aggregated normalised attention coefficients between nodes i and j, across all eight attention heads ($\sum_{k=1}^{K} \alpha_{ij}^{(k)} + \alpha_{ji}^{(k)}$).

brain mesh). To do this, we have leveraged functional MRI data from the Human Connectome Project (HCP). We found that graph convolutional methods (such as GCNs and GATs) are capable of setting state-of-the-art results, exploiting the underlying structure of a brain mesh better than all prior approaches, enabling more informed decisions.

The main qualitative findings of the method (clearly demonstrating improved segmentation outputs on a given test brain) are given in Figure 4.3. In interests of brevity, further details about the experimental setup are held out from this document—for further details,

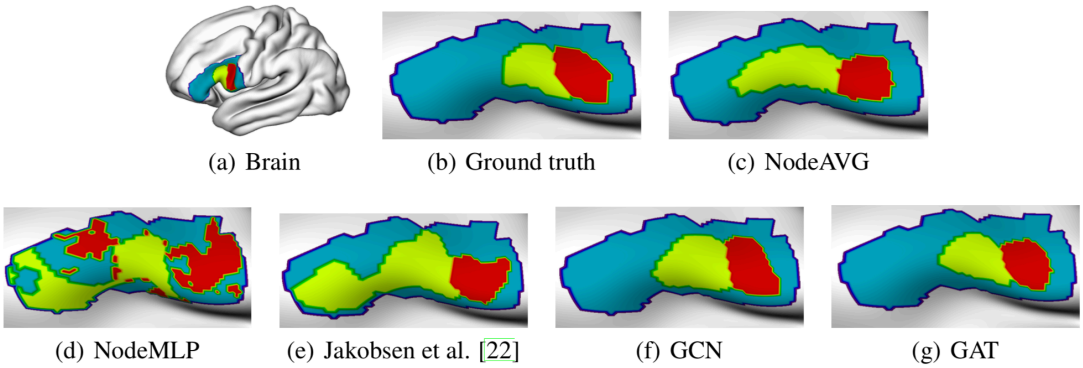


Figure 4.3: Area parcellation qualitative results for several methods on a test subject. The approach of Jakobsen *et al.* [79] is the prior state-of-the-art.

please see [29].

4.6.2 Neural paratope prediction

Antibodies are a critical part of the immune system, having the function of directly neutralising or tagging undesirable objects (the antigens) for future destruction. Here we consider the task of *paratope prediction*: predicting the amino acids of an antibody that participate in binding to a target antigen. A viable paratope predictor is a significant facilitator to antibody design, which in turn will contribute to the development of personalised medicine.

In this work—performed by Andreea Deac as her Part II project (under my supervision)—we build on Parapred, the previous state of the art approach of Liberis *et al.* [108], substituting its convolutional and recurrent layers with à trous convolutional and attentional layers, respectively. These layers have been shown to perform more favourably, and allowed us to, for the first time, positively exploit antigen data. We do this through *cross-modal attention*, allowing amino acids of the antibody to attend over the amino acids of the antigen (which could be seen as a GAT-like model applied to a bipartite antibody-antigen graph). This allowed us to set new state-of-the-art results on this task, along with obtaining insightful interpretations about the model's mechanism of action—some of which may be seen in Figure 4.4.

As above, further details on the model architecture and experimental setup have been omitted in interests of brevity—for further details, please see [32].

4.6.3 Additional related work

Lastly, I will outline several interesting relevant research directions that leverage or further build on GAT(-like) models. The list is by no means exhaustive.

- It is natural to extend the GAT layer to handle *edge features*, as they can be used to further condition the attention mechanism—and, indeed, it is expected that the GAT model will thrive the most in an environment with rich edge features. The RGAT [20] and EAGCN [157] models both extend the GAT to handle *discrete* edge features, while EGAT [52] fully generalises the setup for *adaptive edge features*. In all cases, the recovered GAT architectures compare favourably to several baselines.
- *Gated Attention Networks* (GaAN) [197] introduce *gating mechanisms* into the multi-head attention system of GATs, in order to give different value to different

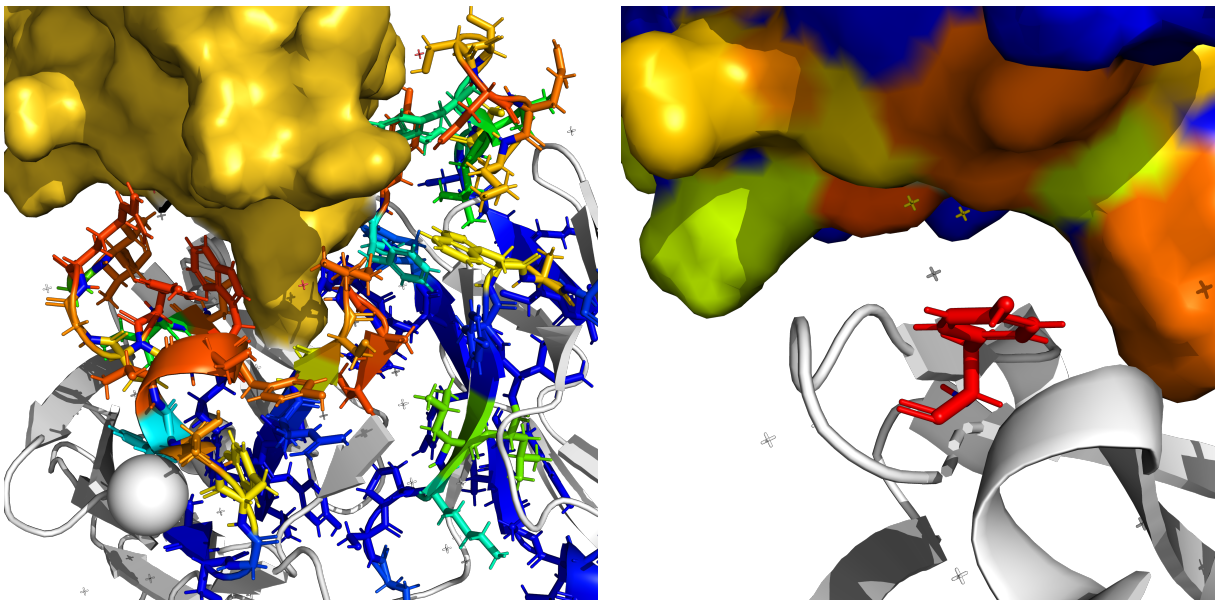


Figure 4.4: For a test antibody-antigen complex: **Left:** Antibody residue binding probabilities to the antigen (in gold) assigned by our paratope predictor [32]. **Right:** Normalised antigen attention weights for a single (binding) antibody residue (in red). Warmer colours indicate higher probabilities/coefficients.

heads' computations. Several challenging baselines are outperformed on both inductive node classification tasks (Reddit, PPI) and a spatiotemporal traffic speed forecasting task (METR-LA).

- *DeepInf* [133] leverages graph convolutional layers to modelling *influence locality* (predicting whether a node will perform a particular action, given the action statuses of its r-hop neighbours at a particular point in time). Notably, this is the first study where attentional mechanisms (GAT) appear to be necessary for surpassing baseline approaches (such as SVMs or logistic regression), given the heterogeneity of the edges. Furthermore, a comprehensive qualitative analysis is performed on the action mechanism of the various attention heads employed by the GAT model.

- In *Attention, Learn to Solve Routing Problems!* [96], GAT-like layers (using the Transformer attention mechanism [173]) have been successfully applied to solving *combinatorial optimisation* problems, specifically the Travelling Salesman Problem.

- *PeerNets* [167] augment a standard convolutional neural network architecture for image classification with GAT-like layers over a graph of "neighbouring" feature maps from related images in a training dataset. This makes the learnt feature maps substantially more robust, providing a strong defence against a variety of adversarial attacks while sacrificing almost no test accuracy.

- Lastly, *hyperbolic attention networks* [62] generalise the attentional mechanism to

act in *hyperbolic space*, making it theoretically more favourable to tasks requiring *hierarchical reasoning* (which is often the case for trees and graphs). Favourable results against a baseline GAT on Cora and Citeseer further compound these insights.

4.7 Summary

In this chapter, I have presented graph attention networks (GATs), novel convolution-style neural networks that operate on graph-structured data, leveraging masked self-attentional layers. The graph attentional layer utilised throughout these networks is computationally efficient (does not require costly matrix operations, and is parallelisable across all nodes in the graph), allows for (implicitly) assigning different importances to different nodes within a neighbourhood while dealing with different sized neighbourhoods, and does not depend on knowing the entire graph structure upfront—thus addressing many of the theoretical issues with prior approaches. Results, both within the original GAT paper [174] and the numerous subsequently published work, highlight the importance of such architectures towards building principled graph convolutional networks.

Chapter 5

Local-global mutual information maximisation on graphs

> "If intelligence is a cake, the *bulk* of the cake is unsupervised learning, the *icing* on the cake is supervised learning, and the *cherry* on the cake is reinforcement learning."
>
> Yann LeCun

As exemplified by the above quote, from the perspective of mimicking biological systems (primarily the *brain*), *unsupervised learning* is the most important learning setup—it requires reasoning about the entirety of the input state without any auxiliary supervision signal, and is how humans actually learn in most cases. As a motivating example, we are capable of extrapolating the complex features that make an entity a *"cat"*—to the level where we can easily detect cats in the wild—with only a small amount of actual occasions (e.g. during our childhood) where a *supervisor* was nearby to instruct us that we were, in fact, looking at a cat. Therefore, advances in unsupervised learning are always poised to have a wide range of immediate effects to the general state-of-the-art in artificial intelligence, and substantial attention must be dedicated to its advancement.

While graph-structured machine learning techniques have seen significant successes recently, most of the prominent methods use a *supervised learning* setup, which is often not possible as most graph data in the wild is unlabelled[1]. In addition, it is often desirable to discover novel or interesting structure from large-scale graphs, and as such, unsupervised graph learning is essential for many important tasks.

[1] In many cases of interest, e.g. prediction on large social networks, the task of interest may not even be *known* at training time.

As a fitting final contribution of this dissertation, I will now restrict my attention solely to the unsupervised learning setup on graphs. Upon highlighting several shortcomings of combining "modern" graph convolutional architectures [49, 92, 174] with traditionally used random walk objectives [61, 131, 169], I will present *mutual information maximisation* as a very promising objective for unsupervised learning on graphs. Through generalising one such approach, *Deep InfoMax* (DIM) [73] to the graph domain, I verify that this is indeed a viable path—one that will be validated through extensive theoretical and practical results throughout this chapter.

The resulting approach, *Deep Graph Infomax* (DGI) [175], is readily applicable to both transductive and inductive learning setups, and it demonstrates competitive performance on a variety of node classification benchmarks, which at times even exceeds the performance of supervised learning.

5.1 Preliminaries

Initially, I will outline several relevant research directions whose efforts are joined together to create DGI—along with a comprehensive survey of related work.

5.1.1 Unsupervised graph learning

Currently, the dominant algorithms for unsupervised representation learning with graph-structured data rely on random walk-based objectives [61, 64, 131, 169], sometimes further simplified to reconstruct adjacency information [38, 91]. The underlying intuition is to train an encoder network so that nodes that are "close" in the input graph are also "close" in the representation space.

While powerful—and related to traditional metrics such as the personalised PageRank score [81]—random walk methods suffer from known limitations. Most prominently, the random-walk objective is known to over-emphasise proximity information at the expense of structural information [137], and performance is highly dependent on hyperparameter choice [61, 131]. Moreover, with the introduction of stronger encoder models based on graph convolutions [49], it is unclear whether random-walk objectives actually provide any useful signal, as these encoders already enforce an inductive bias that neighbouring nodes have similar representations.

5.1.2 Mutual information estimation and maximisation

Through DGI, I propose an alternative objective for unsupervised graph learning that is based on *mutual information*, rather than random walks. Recently, scalable estimation of mutual information was made both possible and practical through Mutual Information Neural Estimation [9], which relies on training a *statistics network* as a classifier of samples coming from the joint distribution of two random variables and their product of marginals. Following on MINE, [73] introduced Deep InfoMax (DIM) for learning representations of high-dimensional data. DIM trains an encoder model to maximise the mutual information between a high-level "global" representation and "local" parts of the input (such as patches of an image). This encourages the encoder to carry the type of information that is present in all locations (and thus is *globally relevant*), such as would be the case of a class label.

DIM relies heavily on convolutional neural network structure in the context of image data, and to my knowledge, no work has applied mutual information maximisation to graph-structured inputs. Here, I adapt ideas from DIM to the graph domain, which can be thought of as having a more general type of structure than the ones captured by convolutional neural networks.

5.1.3 Related work

Contrastive methods An important approach for unsupervised learning of representations is to train an encoder to be *contrastive* between representations that capture statistical dependencies of interest and those that do not. For example, a contrastive approach may employ a *scoring function*, training the encoder to increase the score on "real" input (a.k.a. positive examples) and decrease the score on "fake" input (a.k.a. negative samples). Contrastive methods are central to many popular word-embedding methods [27, 116, 119], but they are found in many unsupervised algorithms for learning representations of graph-structured input as well. There are many ways to score a representation, but in the graph literature the most common techniques use classification [61, 65, 91, 131], though other scoring functions are used [13, 38]. DGI is also contrastive in this respect, as our objective is based on classifying local-global pairs and negative-sampled counterparts.

Sampling strategies A key implementation detail to contrastive methods is how to draw positive and negative samples. The prior work above on unsupervised graph representation learning relies on a local contrastive loss (enforcing proximal nodes to have similar embeddings). Positive samples typically correspond to pairs of nodes that appear together

within *short random walks* in the graph—from a language modelling perspective, effectively treating nodes as *words* and random walks as *sentences*. Recent work by [13] uses node-anchored sampling as an alternative. The negative sampling for these methods is primarily based on sampling of random pairs, with recent work adapting this approach to use a curriculum-based negative sampling scheme, with progressively "closer" negative examples [191] or introducing an adversary to select the negative examples [14].

Predictive coding Contrastive predictive coding (CPC) [128] is another method for learning deep representations based on mutual information maximisation. Like the models above, CPC is also contrastive, in this case using an estimate of the conditional density, in the form of noise contrastive estimation [63] as the scoring function. However, unlike my approach, CPC and the graph methods above are all *predictive*: the contrastive objective effectively trains a predictor between structurally-specified parts of the input (e.g. between neighbouring node pairs or between a node and its neighbourhood). Our approach differs in that we contrast global/local parts of a graph simultaneously, where the global variable is computed from all local variables.

To the best of my knowledge, the sole prior works that instead focus on contrasting "global" and "local" representations on graphs do so via (auto-)encoding objectives on the adjacency matrix [180] and incorporation of community-level constraints into node embeddings [182]. Both methods rely on matrix factorisation-style losses and are thus not scalable to larger graphs.

5.2 DGI methodology

Now I will present the Deep Graph Infomax method in a top-down fashion: starting with an abstract overview of the specific unsupervised learning setup, followed by an exposition of the objective function optimised by the method, and concluding by enumerating all the steps of the procedure in a single-graph setting.

5.2.1 Graph-based unsupervised learning setup

I assume a generic graph-based unsupervised machine learning setup: we are provided with a set of *node features*, $\mathbf{X} = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\}$, where N is the number of nodes in the graph and $\vec{x}_i \in \mathbb{R}^F$ represents the features of node i. We are also provided with relational information between these nodes in the form of an *adjacency matrix*, $\mathbf{A} \in \mathbb{R}^{N \times N}$. While $\mathbf{A}$ may consist of arbitrary real numbers (or even arbitrary edge features), in all experiments I will assume the graphs to be *unweighted*, i.e. $A_{ij} = 1$ if there exists an edge $i \to j$ in the graph and $A_{ij} = 0$ otherwise.

The objective is to learn an *encoder*, $\mathcal{E} : \mathbb{R}^{N \times F} \times \mathbb{R}^{N \times N} \to \mathbb{R}^{N \times F'}$, such that $\mathcal{E}(\mathbf{X}, \mathbf{A}) = \mathbf{H} = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}$ represents high-level representations $\vec{h}_i \in \mathbb{R}^{F'}$ for each node i. These representations may then be used for downstream tasks, such as node classification.

Here the focus will be on *graph convolutional* encoders—a flexible class of node embedding architectures, which generate node representations by repeated aggregation over local node neighbourhoods [49]. A key consequence is that the produced node embeddings, $\vec{h}_i$, *summarise a patch* of the graph centered around node i rather than just the node itself. In what follows, I will often refer to $\vec{h}_i$ as *patch representations* to emphasise this point.

5.2.2 Local-global mutual information maximisation

DGI's approach to learning the encoder relies on *maximising local mutual information*—that is, I seek to obtain node (i.e. local) representations that capture the global information content of the entire graph, represented by a *summary vector*, $\vec{s}$.

In order to obtain the graph-level summary vectors, $\vec{s}$, I leverage a *readout function*, $\mathcal{R} : \mathbb{R}^{N \times F} \to \mathbb{R}^{F}$, and use it to summarise the obtained patch representations into a *graph-level representation*; i.e., $\vec{s} = \mathcal{R}(\mathcal{E}(\mathbf{X}, \mathbf{A}))$.

As a proxy for maximising the local mutual information, I employ a *discriminator*, $\mathcal{D} : \mathbb{R}^{F} \times \mathbb{R}^{F} \to \mathbb{R}$, such that $\mathcal{D}(\vec{h}_i, \vec{s})$ represents the probability scores assigned to this patch-summary pair (should be higher for patches contained within the summary).

Negative samples for $\mathcal{D}$ are provided by pairing the summary $\vec{s}$ from $(\mathbf{X}, \mathbf{A})$ with patch representations $\vec{\widetilde{h}}_j$ of an alternative graph, $(\widetilde{\mathbf{X}}, \widetilde{\mathbf{A}})$. In a multi-graph setting, such graphs may be obtained as other elements of a training set. However, for a single graph, an explicit (stochastic) *corruption function*, $\mathcal{C} : \mathbb{R}^{N \times F} \times \mathbb{R}^{N \times N} \to \mathbb{R}^{M \times F} \times \mathbb{R}^{M \times M}$, is required to obtain a negative example from the original graph, i.e. $(\widetilde{\mathbf{X}}, \widetilde{\mathbf{A}}) = \mathcal{C}(\mathbf{X}, \mathbf{A})$. The choice of the negative sampling procedure will govern the specific kinds of structural information that is desirable to be captured as a byproduct of this maximisation.

For the objective, I follow the intuitions from Deep InfoMax [73] and use a noise-contrastive type objective with a standard binary cross-entropy (BCE) loss between the samples from the joint (positive examples) and the product of marginals (negative examples). Following

their work, I use the following objective[2]:

$$\mathcal{L} = \frac{1}{N+M}\mathbb{E}_{(\mathbf{X},\mathbf{A})}\left[\sum_{i=1}^{N}\log\mathcal{D}\left(\vec{h}_i,\vec{s}\right)+\mathbb{E}_{(\widetilde{\mathbf{X}},\widetilde{\mathbf{A}})}\left[\sum_{j=1}^{M}\log\left(1-\mathcal{D}\left(\vec{\widetilde{h}}_j,\vec{s}\right)\right)\right]\right] \tag{5.1}$$

This approach effectively maximises mutual information between $\vec{h}_i$ and $\vec{s}$, based on the Jensen-Shannon divergence[3] between the joint and the product of marginals.

As all of the derived patch representations are driven to preserve mutual information with the global graph summary, this allows for discovering and preserving similarities on the patch-level—for example, distant nodes with similar structural roles, which are known to be a strong predictor for many node classification tasks [36]. Note that this is a "reversed" version of the argument given by [73]: for node classification, our aim is for the *patches* to establish links to similar patches across the graph, rather than enforcing the summary to contain all of these similarities (however, both of these effects should in principle occur simultaneously).

5.2.3 Theoretical motivation

I now provide some intuition that connects the classification error of the discriminator to mutual information maximisation on graph representations.

Lemma 1. *Let $\{\mathbf{X}^{(k)}\}_{k=1}^{|\mathbf{X}|}$ be a set of node representations drawn from an empirical probability distribution of graphs, $p(\mathbf{X})$, with finite number of elements, $|\mathbf{X}|$, such that $p(\mathbf{X}^{(k)}) = p(\mathbf{X}^{(k')})\ \forall k,k'$. Let $\mathcal{R}(\cdot)$ be a deterministic readout function on graphs and $\vec{s}^{(k)} = \mathcal{R}(\mathbf{X}^{(k)})$ be the summary vector of the k-th graph, with marginal distribution $p(\vec{s})$. The optimal classifier between the joint distribution $p(\mathbf{X},\vec{s})$ and the product of marginals $p(\mathbf{X})p(\vec{s})$, assuming class balance, has an error rate upper bounded by $\mathrm{Err}^* = \frac{1}{2}\sum_{k=1}^{|\mathbf{X}|}p(\vec{s}^{(k)})^2$. This upper bound is achieved if $\mathcal{R}$ is injective.*

Proof. Denote by $\mathcal{Q}^{(k)}$ the set of all graphs in the input set that are mapped to $\vec{s}^{(k)}$ by $\mathcal{R}$, i.e. $\mathcal{Q}^{(k)} = \{\mathbf{X}^{(j)} \mid \mathcal{R}(\mathbf{X}^{(j)}) = \vec{s}^{(k)}\}$. As $\mathcal{R}(\cdot)$ is deterministic, samples from the joint, $(\mathbf{X}^{(k)},\vec{s}^{(k)})$ are drawn from the product of marginals with probability $p(\vec{s}^{(k)})p(\mathbf{X}^{(k)})$, which decomposes into:

$$p(\vec{s}^{(k)})\sum_{\vec{s}}p(\mathbf{X}^{(k)},\vec{s}) = p(\vec{s}^{(k)})p(\mathbf{X}^{(k)}|\vec{s}^{(k)})p(\vec{s}^{(k)}) = \frac{p(\mathbf{X}^{(k)})}{\sum_{\mathbf{X}'\in\mathcal{Q}^{(k)}}p(\mathbf{X}')}p(\vec{s}^{(k)})^2 \tag{5.2}$$

[2] Note that, in [73], a softplus version of the binary cross-entropy is used.

[3] The "GAN" distance defined here—as per [54] and [126]—and Jensen-Shannon divergence can be related by $D_{GAN} = 2D_{JS} - \log 4$. Therefore, any parameters that optimise one also optimise the other.

For convenience, let $\rho^{(k)} = \frac{p(\mathbf{X}^{(k)})}{\sum_{\mathbf{X}' \in \mathcal{Q}^{(k)}} p(\mathbf{X}')}$. As, by definition, $\mathbf{X}^{(k)} \in \mathcal{Q}^{(k)}$, it holds that $\rho^{(k)} \leq 1$. This probability ratio is maximised at 1 when $\mathcal{Q}^{(k)} = \{\mathbf{X}^{(k)}\}$, i.e. when $\mathcal{R}$ is injective for $\mathbf{X}^{(k)}$. The probability of drawing any sample of the joint from the product of marginals is then bounded above by $\sum_{k=1}^{|\mathbf{X}|} p(\vec{s}^{(k)})^2$. As the probability of drawing $(\mathbf{X}^{(k)}, \vec{s}^{(k)})$ from the joint is $\rho^{(k)} p(\vec{s}^{(k)}) \geq \rho^{(k)} p(\vec{s}^{(k)})^2$, we know that classifying these samples as coming from the joint has a lower error than classifying them as coming from the product of marginals. The error rate of such a classifier is then the probability of drawing a sample from the joint as a sample from product of marginals under the mixture probability, which we can bound by $\text{Err} \leq \frac{1}{2} \sum_{k=1}^{|\mathbf{X}|} p(\vec{s}^{(k)})^2$, with the upper bound achieved, as above, when $\mathcal{R}(\cdot)$ is injective for all elements of $\{\mathbf{X}^{(k)}\}$. □

It may be useful to note that $\frac{1}{2|\mathbf{X}|} \leq \text{Err}^* \leq \frac{1}{2}$. The first result is obtained via a trivial application of Jensen's inequality, while the other extreme is reached only in the edge case of a constant readout function (when every example from the joint is also an example from the product of marginals, so no classifier performs better than chance).

Corollary 1. *From now on, assume that the readout function used, $\mathcal{R}$, is injective. Assume the number of allowable states in the space of $\vec{s}$, $|\vec{s}|$, is greater than or equal to $|\mathbf{X}|$. Then, for $\vec{s}^\star$, the optimal summary under the classification error of an optimal classifier between the joint and the product of marginals, it holds that $|\vec{s}^\star| = |\mathbf{X}|$.*

Proof. By injectivity of $\mathcal{R}$, we know that $\vec{s}^\star = \text{argmin}_{\vec{s}} \text{Err}^*$. As the upper error bound, Err^*, is a simple geometric sum, we know that this is minimised when $p(\vec{s}^{(k)})$ is uniform. As $\mathcal{R}(\cdot)$ is deterministic, this implies that each potential summary state would need to be used at least once. Combined with the condition $|\vec{s}| \geq |\mathbf{X}|$, we conclude that the optimum has $|\vec{s}^\star| = |\mathbf{X}|$. □

Theorem 1. $\vec{s}^\star = \text{argmax}_{\vec{s}} \text{MI}(\mathbf{X}; \vec{s})$*, where* MI *is mutual information.*

Proof. This follows from the fact that the mutual information is invariant under invertible transforms. As $|\vec{s}^\star| = |\mathbf{X}|$ and $\mathcal{R}$ is injective, it has an inverse function, $\mathcal{R}^{-1}$. It follows then that, for any $\vec{s}$, $\text{MI}(\mathbf{X}; \vec{s}) \leq H(\mathbf{X}) = \text{MI}(\mathbf{X}; \mathbf{X}) = \text{MI}(\mathbf{X}; \mathcal{R}(\mathbf{X})) = \text{MI}(\mathbf{X}; \vec{s}^\star)$, where H is entropy. □

Theorem 1 shows that for finite input sets and suitable deterministic functions, minimising the classification error in the discriminator can be used to maximise the mutual information between the input and output. However, as was shown in [73], this objective alone is not enough to learn useful representations. As in their work, I discriminate between the global summary vector and local high-level representations.

Theorem 2. *Let $\mathbf{X}_i^{(k)} = \{\vec{x}_j\}_{j \in n(\mathbf{X}^{(k)}, i)}$ be the neighbourhood of the node i in the k-th graph that collectively maps to its high-level features, $\vec{h}_i = \mathcal{E}(\mathbf{X}_i^{(k)})$, where n is the neighbourhood function that returns the set of neighbourhood indices of node i for graph $\mathbf{X}^{(k)}$,*

and $\mathcal{E}$ is a deterministic encoder function. Let us assume that $|\mathbf{X}_i| = |\mathbf{X}| = |\vec{s}| \geq |\vec{h}_i|$. Then, the $\vec{h}_i$ that minimises the classification error between $p(\vec{h}_i, \vec{s})$ and $p(\vec{h}_i)p(\vec{s})$ also maximises $\text{MI}(\mathbf{X}_i^{(k)}; \vec{h}_i)$.

Proof. Given our assumption of $|\mathbf{X}_i| = |\vec{s}|$, there exists an inverse $\mathbf{X}_i = \mathcal{R}^{-1}(\vec{s})$, and therefore $\vec{h}_i = \mathcal{E}(\mathcal{R}^{-1}(\vec{s}))$, i.e. there exists a deterministic function $(\mathcal{E} \circ \mathcal{R}^{-1})$ mapping $\vec{s}$ to $\vec{h}_i$. The optimal classifier between the joint $p(\vec{h}_i, \vec{s})$ and the product of marginals $p(\vec{h}_i)p(\vec{s})$ then has (by Lemma 1) an error rate upper bound of $\text{Err}^* = \frac{1}{2}\sum_{k=1}^{|\mathbf{X}|} p(\vec{h}_i^{(k)})^2$. Therefore (as in Corollary 1), for the optimal $\vec{h}_i$, $|\vec{h}_i| = |\mathbf{X}_i|$, which by the same arguments as in Theorem 1 maximises the mutual information between the neighbourhood and high-level features, $\text{MI}(\mathbf{X}_i^{(k)}; \vec{h}_i)$. □

This motivates the use of a classifier between samples from the joint and the product of marginals, and using the binary cross-entropy (BCE) loss to optimise this classifier is well-understood in the context of neural network optimisation.

5.2.4 Overview of DGI

Assuming the single-graph setup (i.e., $(\mathbf{X}, \mathbf{A})$ provided as input), we will now summarise the steps of the Deep Graph Infomax procedure:

1. Sample a negative example by using the corruption function: $(\widetilde{\mathbf{X}}, \widetilde{\mathbf{A}}) \sim \mathcal{C}(\mathbf{X}, \mathbf{A})$.
2. Obtain patch representations, $\vec{h}_i$ for the input graph by passing it through the encoder: $\mathbf{H} = \mathcal{E}(\mathbf{X}, \mathbf{A}) = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}$.
3. Obtain patch representations, $\vec{\widetilde{h}}_j$ for the negative example by passing it through the encoder: $\widetilde{\mathbf{H}} = \mathcal{E}(\widetilde{\mathbf{X}}, \widetilde{\mathbf{A}}) = \{\vec{\widetilde{h}}_1, \vec{\widetilde{h}}_2, \ldots, \vec{\widetilde{h}}_M\}$.
4. Summarise the input graph by passing its patch representations through the readout function: $\vec{s} = \mathcal{R}(\mathbf{H})$.
5. Update parameters of $\mathcal{E}$, $\mathcal{R}$ and $\mathcal{D}$ by applying gradient descent to maximise the cross-entropy objective expressed in Equation 5.1.

This algorithm is fully summarised by Figure 5.1.

5.3 Classification performance

I have assessed the benefits of the representation learnt by the DGI encoder on a variety of node classification tasks (transductive as well as inductive), obtaining competitive results. In each case, DGI was used to learn patch representations in a fully unsupervised manner,

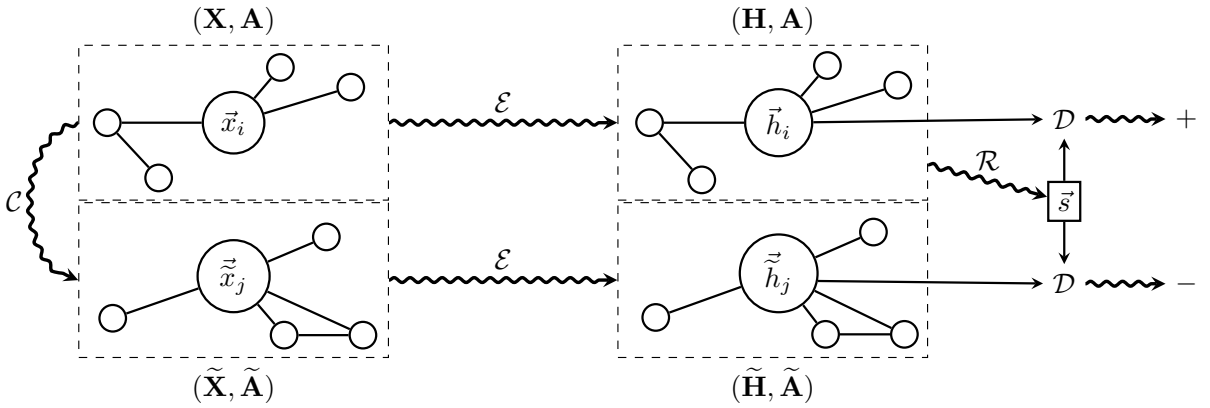


Figure 5.1: A high-level overview of Deep Graph Infomax. Refer to Section 5.2.4 for more details.

Table 5.1: Summary of the datasets used in the DGI experiments.

Dataset	Task	Nodes	Edges	Features	Classes	Train/Val/Test Nodes
Cora	Transductive	2,708	5,429	1,433	7	140/500/1,000
Citeseer	Transductive	3,327	4,732	3,703	6	120/500/1,000
Pubmed	Transductive	19,717	44,338	500	3	60/500/1,000
Reddit	Inductive	231,443	11,606,919	602	41	151,708/23,699/55,334
PPI	Inductive	56,944 (24 graphs)	818,716	50	121 (multilbl.)	44,906/6,514/5,524 (20/2/2 graphs)

followed by evaluating the node-level classification utility of these representations. This was performed by directly using these representations to train and test a simple linear (logistic regression) classifier.

5.3.1 Datasets

I follow the experimental setup described in [92] and [64] on the following benchmark tasks: (1) classifying research papers into topics on the Cora, Citeseer and Pubmed citation networks [156]; (2) predicting the community structure of a social network modelled with Reddit posts; and (3) classifying protein roles within protein-protein interaction (PPI) networks [200], requiring generalisation to unseen networks.

All datasets except for Reddit have already been described in Section 4.5.1. Reddit is a large graph dataset (231,443 nodes and 11,606,919 edges) of Reddit posts created during September 2014 (derived and preprocessed as in [64]). The objective is to predict

the posts' community ("*subreddit*"), based on the GloVe embeddings of their content and comments [129], as well as metrics such as score or number of comments. Posts are linked together in the graph if the same user has commented on both. Reusing the inductive setup of [64], posts made in the first 20 days of the month are used for training, while the remaining posts are used for validation or testing and are *invisible* to the training algorithm.

Further information on the datasets may be found in Table 5.1.

5.3.2 Experimental setup

For each of three experimental settings (transductive learning, inductive learning on large graphs, and multiple graphs), I have employed distinct encoders and corruption functions appropriate to that setting (described below).

Transductive learning For the transductive learning tasks (Cora, Citeseer and Pubmed), the encoder is a one-layer Graph Convolutional Network (GCN) model [92], with the following propagation rule:

$$\mathcal{E}(\mathbf{X}, \mathbf{A}) = \sigma\left(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{\Theta}\right) \tag{5.3}$$

where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is the adjacency matrix with inserted self-loops and $\hat{\mathbf{D}}$ is its corresponding degree matrix; i.e. $\hat{D}_{ii} = \sum_j \hat{A}_{ij}$. For the nonlinearity, σ, I have applied the parametric ReLU (PReLU) function [67], and $\mathbf{\Theta} \in \mathbb{R}^{F \times F'}$ is a learnable linear transformation applied to every node, with $F' = 512$ features being computed (specially, $F' = 256$ on Pubmed due to memory limitations).

The corruption function used in this setting is designed to encourage the representations to properly encode structural similarities of different nodes in the graph; for this purpose, $\mathcal{C}$ preserves the original adjacency matrix ($\widetilde{\mathbf{A}} = \mathbf{A}$), whereas the corrupted features, $\widetilde{\mathbf{X}}$, are obtained by row-wise shuffling of $\mathbf{X}$. That is, the corrupted graph consists of exactly the same nodes as the original graph, but they are located in different places in the graph, and will therefore receive different patch representations. I will subsequently demonstrate that DGI is stable to other choices of corruption functions, but I find that those that preserve the graph structure result in the strongest features.

Inductive learning on large graphs For inductive learning, the GCN update rule may no longer be used in the encoder (as the learned filters rely on a fixed and known adjacency matrix); instead, I apply the *mean-pooling* propagation rule, as used by the GraphSAGE-

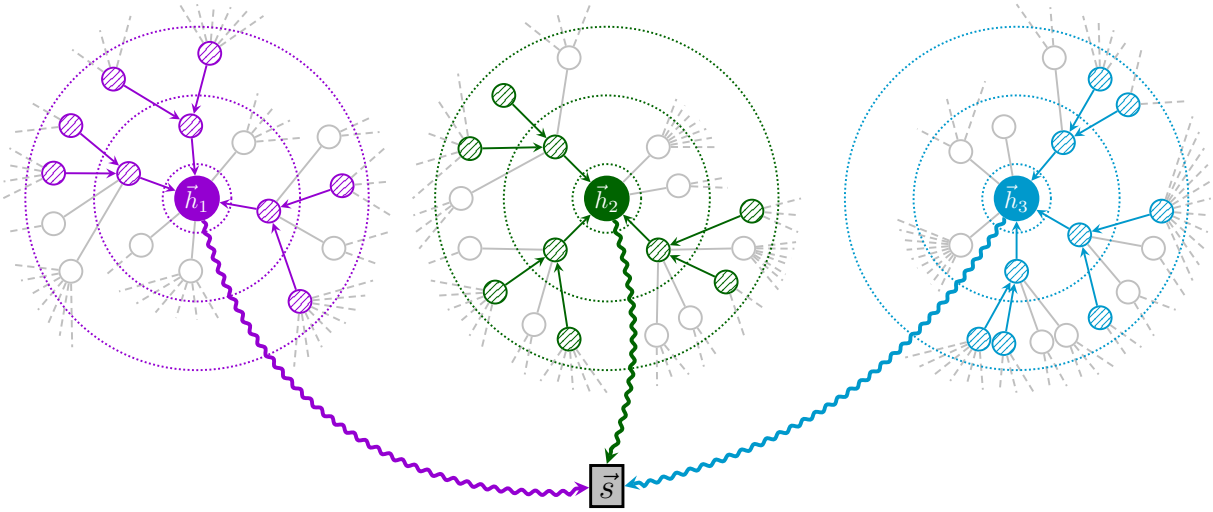


Figure 5.2: The DGI setup on large graphs (such as Reddit). Summary vectors, $\vec{s}$, are obtained by combining several subsampled patch representations, $\vec{h}_i$ (here obtained by sampling three and two neighbours in the first and second level, respectively).

GCN [64] model:

$$\mathrm{MP}(\mathbf{X}, \mathbf{A}) = \hat{\mathbf{D}}^{-1}\hat{\mathbf{A}}\mathbf{X}\boldsymbol{\Theta} \tag{5.4}$$

with parameters defined as in Equation 5.3. Note that multiplying by $\hat{\mathbf{D}}^{-1}$ actually performs a normalised sum (hence the mean-pooling). While Equation 5.4 explicitly specifies the adjacency and degree matrices, *they are not needed*: identical inductive behaviour may be observed by a *constant* attention mechanism across the node's neighbours, as used by the Const-GAT model [174].

For Reddit, the encoder is a three-layer mean-pooling model with skip connections [68]:

$$\widetilde{\mathrm{MP}}(\mathbf{X}, \mathbf{A}) = \sigma\left(\mathbf{X}\boldsymbol{\Theta}' \| \mathrm{MP}(\mathbf{X}, \mathbf{A})\right) \qquad \mathcal{E}(\mathbf{X}, \mathbf{A}) = \widetilde{\mathrm{MP}}_3(\widetilde{\mathrm{MP}}_2(\widetilde{\mathrm{MP}}_1(\mathbf{X}, \mathbf{A}), \mathbf{A}), \mathbf{A}) \tag{5.5}$$

where $\|$ is featurewise concatenation (i.e. the central node and its neighbourhood are handled separately). $F' = 512$ features are computed in each MP layer, with the PReLU activation for σ. Given the large scale of the dataset, it will not fit into GPU memory entirely. Therefore, I use the subsampling approach of [64], where a minibatch of nodes is first selected, and then a subgraph centered around each of them is obtained by *sampling node neighbourhoods with replacement*. Specifically, I sample 10, 10 and 25 neighbours at the first, second and third level, respectively—thus, each subsampled patch has 1 + 10 + 100 + 2500 = 2611 nodes. Only the computations necessary for deriving the central node i's patch representation, $\vec{h}_i$, are performed. These representations are then used to derive the summary vector, $\vec{s}$, for the minibatch (Figure 5.2). I have used minibatches of 256 nodes throughout training.

To define the corruption function in this setting, I use a similar approach as in the transductive tasks, but treat each subsampled patch as a separate graph to be corrupted (i.e. I row-wise shuffle the feature matrices within a subsampled patch). Note that this may very likely cause the central node's features to be swapped out for a sampled neighbour's features, further encouraging diversity in the negative samples. The patch representation obtained in the central node is then submitted to the discriminator.

Inductive learning on multiple graphs For the PPI dataset, inspired by previous successful supervised architectures [174], the encoder is a three-layer mean-pooling model with dense skip connections [68, 76]:

$$\mathbf{H}_1 = \sigma\left(\mathrm{MP}_1(\mathbf{X}, \mathbf{A})\right) \tag{5.6}$$

$$\mathbf{H}_2 = \sigma\left(\mathrm{MP}_2(\mathbf{H}_1 + \mathbf{X}\mathbf{W}_{\text{skip}}, \mathbf{A})\right) \tag{5.7}$$

$$\mathcal{E}(\mathbf{X}, \mathbf{A}) = \sigma\left(\mathrm{MP}_3(\mathbf{H}_2 + \mathbf{H}_1 + \mathbf{X}\mathbf{W}_{\text{skip}}, \mathbf{A})\right) \tag{5.8}$$

where $\mathbf{W}_{\text{skip}}$ is a learnable projection matrix, and MP is as defined in Equation 5.4. I compute $F' = 512$ features in each MP layer, using the PReLU activation for σ.

In this multiple-graph setting, I opted to use *randomly sampled training graphs* as negative examples (i.e. the corruption function simply samples a different graph from the training set). I have found this method to be the most stable, considering that over 40% of the nodes have all-zero features in this dataset. To further expand the pool of negative examples, I also apply dropout [162] to the input features of the sampled graph. I have found it beneficial to standardise the learnt embeddings across the training set prior to providing them to the logistic regression model.

Readout, discriminator, and additional training details Across all three experimental settings, I have employed identical readout functions and discriminator architectures.

For the readout function, I use a simple averaging of all the nodes' features:

$$\mathcal{R}(\mathbf{H}) = \sigma\left(\frac{1}{N}\sum_{i=1}^{N}\vec{h}_i\right) \tag{5.9}$$

where σ is the logistic sigmoid nonlinearity. While I have found this readout to perform the best across all our experiments, it is assumed that its power will diminish with the increase in graph size, and in those cases, more sophisticated readout architectures such as set2vec [179] or DiffPool [192] are likely to be more appropriate.

The discriminator scores summary-patch representation pairs by applying a simple bilinear scoring function (similar to the scoring used by [128]):

$$\mathcal{D}(\vec{h}_i, \vec{s}) = \sigma\left(\vec{h}_i^T \mathbf{W} \vec{s}\right) \tag{5.10}$$

Here, $\mathbf{W}$ is a learnable scoring matrix and σ is the logistic sigmoid nonlinearity, used to convert scores into probabilities of $(\vec{h}_i, \vec{s})$ being a positive example.

All models are initialised using Xavier initialisation [50] and trained to maximise the mutual information provided in Equation 5.1 on the available nodes (all nodes for the transductive, and training nodes only in the inductive setup) using the Adam SGD optimizer [88] with an initial learning rate of 0.001 (specially, 10^{-5} on Reddit). On the transductive datasets, I use an early stopping strategy on the observed *training* loss, with a patience of 20 epochs. On the inductive datasets, I train for a fixed number of epochs (150 on Reddit, 20 on PPI).

5.3.3 Results

The results of the comparative evaluation experiments are summarised in Table 5.2.

For the transductive tasks, I report the mean classification accuracy (with standard deviation) on the test nodes of the DGI method after 50 runs of training (followed by logistic regression), and reuse the metrics already reported in [92] for the performance of DeepWalk and GCN, as well as Label Propagation (LP) [198] and Planetoid [190]—a representative supervised random walk method. Specially, I provide results for training the logistic regression on raw input features, as well as DeepWalk with the input features concatenated.

For the inductive tasks, I report the micro-averaged F_1 score on the (unseen) test nodes, averaged after 50 runs of training, and reuse the metrics already reported in [64] for the other techniques. Specifically, as the setup is unsupervised, I compare against the unsupervised GraphSAGE approaches. I also provide supervised results for two related architectures—FastGCN [22] and Avg. pooling [197].

The results demonstrate strong performance being achieved across all five datasets. I particularly note that the DGI approach is competitive with the results reported for the GCN model *with the supervised loss*, even exceeding its performance on the Cora and Citeseer datasets. It is assumed that these benefits stem from the fact that, indirectly, the DGI approach allows for every node to have access to structural properties of the entire graph,

Table 5.2: Summary of results in terms of classification accuracies (on transductive tasks) or micro-averaged F_1 scores (on inductive tasks). In the first column, I highlight the kind of data available to each method during training (**X**: features, **A**: adjacency matrix, **Y**: labels). "GCN" corresponds to a two-layer DGI encoder trained in a supervised manner.

Transductive

Available data	Method	Cora	Citeseer	Pubmed
X	Raw features	$47.9 \pm 0.4\%$	$49.3 \pm 0.2\%$	$69.1 \pm 0.3\%$
A, **Y**	LP [198]	68.0%	45.3%	63.0%
A	DeepWalk [131]	67.2%	43.2%	65.3%
X, **A**	DeepWalk + features	$70.7 \pm 0.6\%$	$51.4 \pm 0.5\%$	$74.3 \pm 0.9\%$
X, **A**	Random-Init (ours)	$69.3 \pm 1.4\%$	$61.9 \pm 1.6\%$	$69.6 \pm 1.9\%$
X, **A**	**DGI** (ours)	**82.3** $\pm 0.6\%$	**71.8** $\pm 0.7\%$	**76.8** $\pm 0.6\%$
X, **A**, **Y**	GCN [92]	81.5%	70.3%	79.0%
X, **A**, **Y**	Planetoid [190]	75.7%	64.7%	77.2%

Inductive

Available data	Method	Reddit	PPI
X	Raw features	0.585	0.422
A	DeepWalk [131]	0.324	—
X, **A**	DeepWalk + features	0.691	—
X, **A**	GraphSAGE-GCN [64]	0.908	0.465
X, **A**	GraphSAGE-mean [64]	0.897	0.486
X, **A**	GraphSAGE-LSTM [64]	0.907	0.482
X, **A**	GraphSAGE-pool [64]	0.892	0.502
X, **A**	Random-Init (ours)	0.933 ± 0.001	0.626 ± 0.002
X, **A**	**DGI** (ours)	**0.940** ± 0.001	**0.638** ± 0.002
X, **A**, **Y**	FastGCN [22]	0.937	—
X, **A**, **Y**	Avg. pooling [197]	0.958 ± 0.001	0.969 ± 0.002

whereas the supervised GCN is limited to only two-layer neighbourhoods (by the extreme sparsity of the training signal and the corresponding threat of overfitting). It should be noted that, while DGI is capable of outperforming equivalent supervised encoder architectures, this performance still does not surpass the current supervised transductive state of the art (which is held by methods such as GraphSGAN [35]). It can further be observed that the DGI method successfully outperformed all the competing unsupervised GraphSAGE approaches on the Reddit and PPI datasets—thus verifying the potential of methods based on local mutual information maximisation in the inductive node classification domain. The Reddit results are competitive with the supervised state of the art, whereas on PPI the gap is still large—I believe this can be attributed to the extreme sparsity of available node features (over 40% of the nodes having all-zero features), that our encoder heavily relies on.

I note that a *randomly initialised* graph convolutional network may already extract highly useful features and represents a strong baseline—a well-known fact, considering its links to the Weisfeiler-Lehman graph isomorphism test [185], that have already been highlighted and analysed by [92] and [64]. As such, I also provide, as *Random-Init*, the logistic regression performance on embeddings obtained from a randomly initialised encoder. Besides demonstrating that DGI is able to further improve on this strong baseline, it particularly reveals that, on the inductive datasets, previous random walk-based negative sampling methods may have been ineffective in conjunction with a GCN-based encoder for learning appropriate features for the classification task.

Lastly, It should be noted that deeper encoders correspond to more pronounced *mixing* between recovered patch representations, reducing the effective variability of the positive/negative examples' pool. I believe that this is the reason why shallower architectures performed better on some of the datasets. While it cannot be said that these trends will hold in general, with the DGI loss function I generally found benefits from employing *wider*, rather than *deeper* models.

5.4 Qualitative analysis

To compound the quantitative results, I have performed a diverse set of analyses on the embeddings learnt by the DGI algorithm in order to better understand the properties of DGI. I focus the analysis exclusively on the Cora dataset (as it has the smallest number of nodes, significantly aiding clarity).

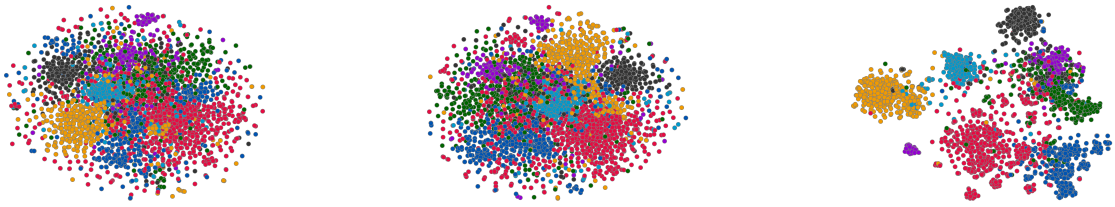


Figure 5.3: t-SNE embeddings of the nodes in the Cora dataset from the raw features (**left**), features from a randomly initialised DGI model (**middle**), and a learned DGI model (**right**). The clusters of the learned DGI model's embeddings are clearly defined, with a Silhouette score of 0.234.

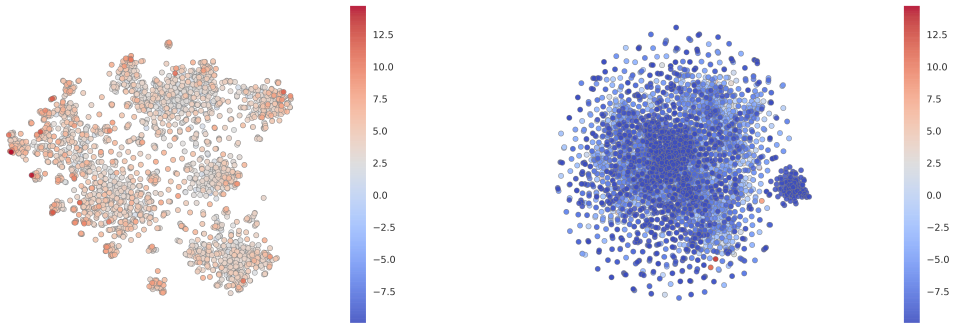


Figure 5.4: Discriminator scores, $\mathcal{D}\left(\vec{h}_i, \vec{s}\right)$, attributed to each node in the Cora dataset shown over a t-SNE of the DGI algorithm. Shown for both the original graph (**left**) and a negative sample (**right**).

5.4.1 Embedding space visualisation

A standard set of "evolving" t-SNE plots [113] of the embeddings is given in Figure 5.3. As expected given the quantitative results, the learnt embeddings' 2D projections exhibit discernible clustering in the 2D projected space (especially compared to the raw features and Random-Init), which respects the seven topic classes of Cora. The projection obtains a Silhouette score [141] of 0.234, which compares favourably with the previous reported score of 0.158 for Embedding Propagation [38].

5.4.2 DGI learning mechanism

I have ran further analyses, revealing insights into DGI's mechanism of learning, isolating *biased* embedding dimensions for pushing the negative example scores down and using the remainder to encode useful information about positive examples. I then leveraged these insights to retain competitive performance to the supervised GCN even after *half* the dimensions are removed from the patch representations provided by the encoder.

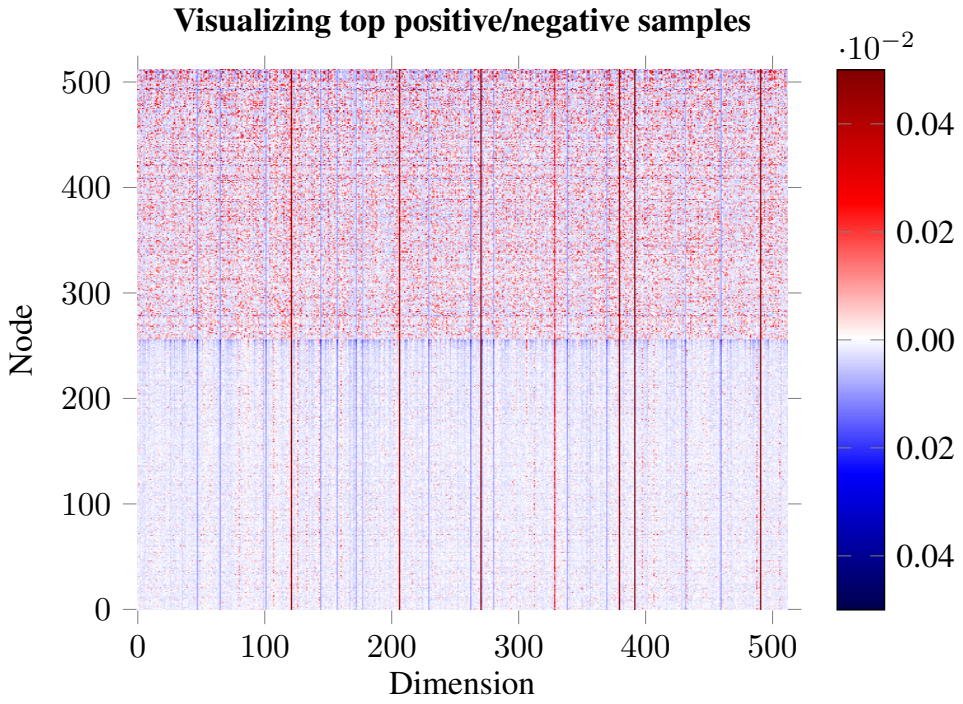


Figure 5.5: The learnt embeddings of the highest-scored positive examples (*upper half*), and the lowest-scored negative examples (*lower half*).

Visualising discriminator scores After obtaining the t-SNE visualisations, attention has turned to the discriminator—and I have visualised the scores it attached to various nodes, for both the positive and a (randomly sampled) negative example (Figure 5.4). From here, an interesting observation can be made—within the "clusters" of the learnt embeddings on the positive Cora graph, only a handful of "hot" nodes are selected to receive high discriminator scores. This suggests that there may be a clear distinction between embedding dimensions used for discrimination and classification, which I more thoroughly investigate in the next paragraph. In addition, it may be observed that, as expected, the model is unable to find any strong structure within a negative example. Lastly, a few negative examples achieve high discriminator scores—a phenomenon caused by the existence of low-degree nodes in Cora (making the probability of a node ending up in an identical context it had in the positive graph non-negligible).

Impact and role of embedding dimensions Guided by the previous result, I have visualised the embeddings for the top-scoring positive and negative examples (Figure 5.5). The analysis revealed existence of distinct dimensions in which both the positive and negative examples are *strongly biased*. I hypothesise that, given the random shuffling, the average *expected* activation of a negative example is zero, and therefore strong biases are required to "push" the example down in the discriminator. The positive examples may then use the remaining dimensions to both counteract this bias and encode patch

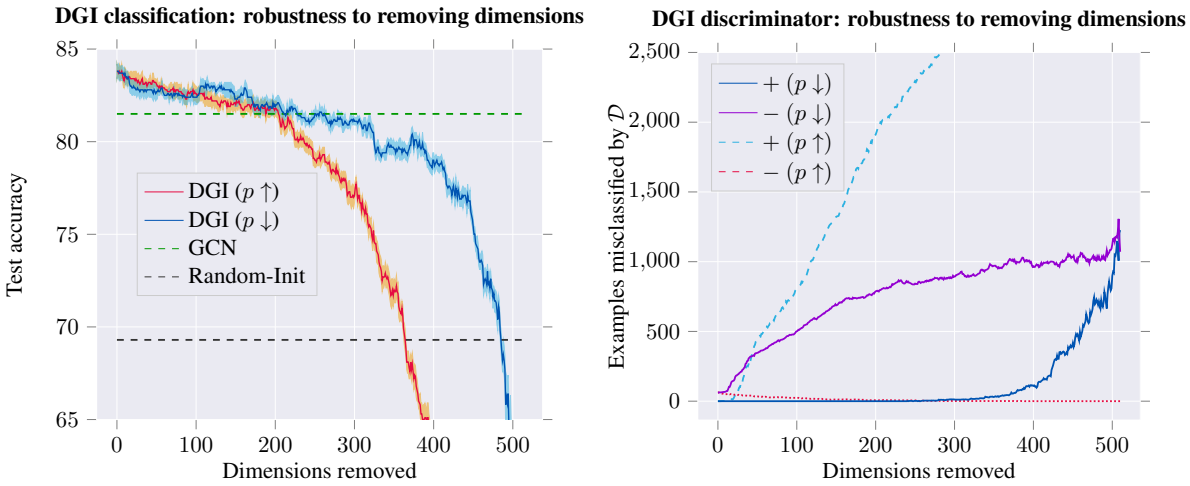


Figure 5.6: Classification performance (in terms of test accuracy of logistic regression; **left**) and discriminator performance (in terms of number of poorly discriminated positive/negative examples; **right**) on the learnt DGI embeddings, after removing a certain number of dimensions from the embedding—either starting with most distinguishing ($p \uparrow$) or least distinguishing ($p \downarrow$).

similarity. To substantiate this claim, I have ordered the 512 dimensions based on how distinguishable the positive and negative examples are in them (using p-values obtained from a t-test as a proxy). I then remove these dimensions from the embedding, respecting this order—either starting from the most distinguishable ($p \uparrow$) or least distinguishable dimensions ($p \downarrow$)—monitoring how this affects both classification and discriminator performance (Figure 5.6). The observed trends largely support my hypothesis: if we start by removing the biased dimensions first ($p \downarrow$), the classification performance holds up for much longer (allowing for removing over *half* of the embedding dimensions while remaining competitive to the supervised GCN), and the positive examples mostly remain correctly discriminated until well over half the dimensions are removed.

5.5 Corruption function robustness

To conclude this chapter, I consider alternatives to the corruption function, $\mathcal{C}$, used to produce negative graphs. I generally find that, for the node classification task, DGI is stable and robust to different strategies. However, for learning graph features towards other kinds of tasks, the design of appropriate corruption strategies remains an area of open research.

The corruption function described in Section 5.3.2 preserves the original adjacency matrix ($\widetilde{\mathbf{A}} = \mathbf{A}$) but corrupts the features, $\widetilde{\mathbf{X}}$, via row-wise shuffling of $\mathbf{X}$. In this case, the negative graph is constrained to be isomorphic to the positive graph, which should not

have to be mandatory. One can instead produce a negative graph by directly *corrupting* the adjacency matrix.

Therefore, I first consider an alternative corruption function $\mathcal{C}$ which preserves the features ($\widetilde{\mathbf{X}} = \mathbf{X}$) but instead adds or removes edges from the adjacency matrix ($\widetilde{\mathbf{A}} \neq \mathbf{A}$). This is done by sampling, i.i.d., a *switch* parameter $\mathbf{\Sigma}_{ij}$, which determines whether to corrupt the adjacency matrix at position (i, j). Assuming a given *corruption rate*, ρ, $\mathcal{C}$ may be defined as performing the following operations:

$$\mathbf{\Sigma}_{ij} \sim \text{Bernoulli}(\rho) \tag{5.11}$$

$$\widetilde{\mathbf{A}} = \mathbf{A} \oplus \mathbf{\Sigma} \tag{5.12}$$

where $\oplus$ is the XOR (exclusive OR) operation.

This alternative strategy produces a negative graph with the same features, but different connectivity. Here, the corruption rate of $\rho = 0$ corresponds to an unchanged adjacency matrix (i.e. the positive and negative graphs are *identical* in this case). In this regime, learning is impossible for the discriminator, and the performance of DGI is in line with a randomly initialised DGI model. At higher rates of noise, however, DGI produces competitive embeddings.

I have also considered *simultaneous* feature shuffling ($\widetilde{\mathbf{X}} \neq \mathbf{X}$) and adjacency matrix perturbation ($\widetilde{\mathbf{A}} \neq \mathbf{A}$), both as described before. I find that DGI still learns useful features under this compound corruption strategy—as expected, given that feature shuffling is already equivalent to an (isomorphic) adjacency matrix perturbation.

From both studies, it may be observed that a certain lower bound on the positive graph perturbation rate is required to obtain competitive node embeddings for the classification task on Cora. Furthermore, the features learned for downstream node classification tasks are most powerful when the negative graph has similar levels of connectivity to the positive graph.

The classification performance peaks when the graph is perturbed to a reasonably high level, but remains *sparse*; i.e. the mixing between the separate 1-step patches is not substantial, and therefore the pool of negative examples is still *diverse* enough. Classification performance is impacted only marginally at higher rates of corruption—corresponding to *dense* negative graphs, and thus a less rich negative example pool—but still considerably outperforming the unsupervised baselines considered here. This could be seen as fur-

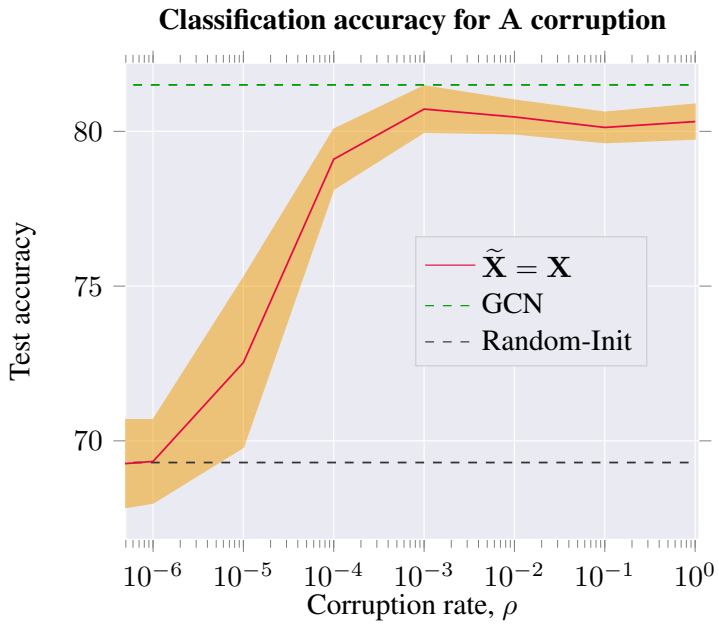


Figure 5.7: DGI also works under a corruption function that modifies only the adjacency matrix ($\widetilde{\mathbf{A}} \neq \mathbf{A}$) on the Cora dataset. The left range ($\rho \to 0$) corresponds to no modifications of the adjacency matrix—therein, performance approaches that of the randomly initialised DGI model. As ρ increases, DGI produces more useful features, but ultimately fails to outperform the feature-shuffling corruption function. **N.B.** log scale used for ρ.

ther motivation for relying solely on feature shuffling, without adjacency perturbations—given that feature shuffling is a trivial way to guarantee a diverse set of negative examples, without incurring significant computational costs per epoch.

The results of this study are visualised in Figures 5.7 and 5.8.

5.6 Summary

In this chapter, I have presented Deep Graph Infomax (DGI), a new approach for learning unsupervised representations on graph-structured data. By leveraging local mutual information maximisation across the graph's patch representations—obtained by powerful graph convolutional architectures—I have been able to obtain node embeddings that are mindful of the global structural properties of the graph. This enables competitive performance across a variety of both transductive and inductive classification tasks, at times even outperforming relevant *supervised* architectures.

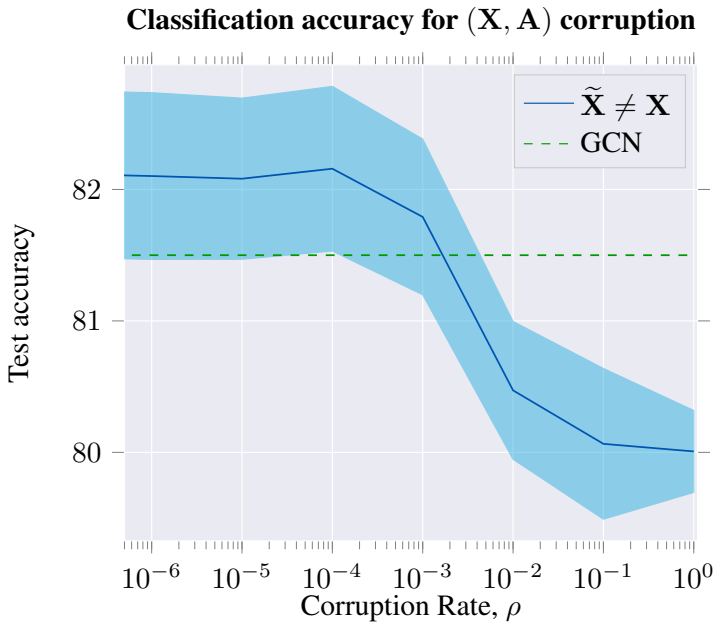


Figure 5.8: DGI is stable and robust under a corruption function that modifies *both* the feature matrix ($\mathbf{X} \neq \widetilde{\mathbf{X}}$) and the adjacency matrix ($\widetilde{\mathbf{A}} \neq \mathbf{A}$) on the Cora dataset. Corruption functions that preserve sparsity ($\rho \approx \frac{1}{N}$) perform the best. However, DGI still performs well even with large disruptions (where edges are added or removed with probabilities approaching 1). **N.B.** log scale used for ρ.

Chapter 6

Conclusions

> "Seldom do more than a few of nature's secrets give way at one time. It will be all too easy for our somewhat artificial prosperity to collapse overnight when it is realised that the use of a few exciting words like *information*, *entropy*, *redundancy*, do not solve all our problems."
>
> Claude E. Shannon

This dissertation has investigated the effects of introducing bespoke **structural inductive biases** into a series of popular neural network architectures and optimisation algorithms. The primary argument posed here—that such bespoke solutions are *necessary* whenever the problem setup or data environment is not optimal for *"standard"* application of deep learning, has been consistently validated across its chapters.

I will conclude the writeup by briefly summarising the key contributions once more, followed by an overview of one of the most important avenues where further work in the area should be deployed.

6.1 Summary of contributions

- In Chapter 3, I have pioneered two *early fusion* architectures for cross-modal learning on homogeneous modalities: the **X-CNN** [176] for images and **X-LSTM** [177] for sequential data. In both cases, it has been shown that these architectures can bring tangible benefits, especially under challenging data scenarios (such as small sample sizes or missing observations). Furthermore, despite the perceived difficulty

of optimising such an architecture's hyperparameters, as well as generalising it to heterogeneous scenarios, it has been found that both of these challenges can be managed; primarily through the **Xsertion** library, developed by Laurynas Karazija [87], and the **XFlow** architecture, developed by Cătălina Cangea [21]—both under my co-supervision.

- In Chapter 4, I have outlined the required theoretical properties that a *graph convolutional layer* should have. Upon concluding that none of the previous experimentally validated architectures have met all of this criteria, I have proposed **Graph Attention Networks** (GATs) [174] as the first operator to do so. GAT's reliance on *masked self-attentional layers* makes it both theoretically powerful and scalable, which has been confirmed by both quantitative and qualitative analyses. The layer is now commonly used within graph neural network architectures, prominently featuring in many successful follow-up studies, including two that I've personally been involved in: the work of Guillem Cucurull on *mesh-based cortical parcellation* [29] and the work of Andreea Deac on *neural paratope prediction* [32] (performed under my direct supervision).

- In Chapter 5, I have focussed on the problem of *unsupervised graph representation learning*, realising that the traditionally used random walk objectives are *incompatible* with the recently proposed graph convolutional network encoders. As a remedy, I have adopted the approach of *local-global mutual information maximisation* which has seen success in the image domain, and generalised it to the graph-structured domain. This contribution is embodied in the **Deep Graph Infomax** (DGI) [175] method, which successfully outperformed many strong unsupervised baselines, and in some cases even the *supervised* variant of its encoder. Substantial qualitative studies verify the method's robustness, discriminative power, and provide further insights into its learning mechanism. I believe that this is a highly promising approach for generic representation learning, and am hopeful to see it extended in the future to more generic graph-structured inputs (such as *spatiotemporal graphs*, allowing for dynamically modifying the node properties through time).

6.2 Structural inductive biases meet *reinforcement learning*

It is obvious that the generality of exploiting domain-specific *structure* readily lends itself to a vast amount of avenues for future work. As such, I choose not to make a comprehensive overview of these avenues at the very end of this dissertation. Rather, I prefer to focus my attention on only one such avenue, where not only I believe such methods can readily thrive, but also one that I hope to personally contribute to over the years follow-

ing the submission of this dissertation.

Namely, it concerns the marriage between structural inductive biases (especially on *graph-structured data*) and **reinforcement learning**. Reinforcement learning has been a critical component in many of deep learning's largest successes [120, 158], but it represents a highly challenging environment for algorithms to learn in. Often, large quantities of input must be consumed before *any* useful reward signal is observed. Therefore, I argue that, perhaps especially in this setting, useful inductive biases can and should readily be exploited. A few such avenues, especially in relation to the methods covered in this dissertation, can be summarised by the following:

- Enabling processing of *cross-modal inputs*. Often, the observations given to us by an RL environment can be multimodal—although most of the ongoing work leveraging environments such as the OpenAI Gym [15] inherently *discard* all but the visual one. This is unlikely to be optimal even for the *videogame* setting, as often very useful cues are given to the player about the obtained reward (or proximity to the reward) in aural form. *Cross-modal architectures*—such as the early-fusion ones described in Chapter 3—may then be leveraged to accomplish efficient usage of such additional signals, accelerating the learning procedure.

- Various aspects of the reinforcement learning *state* may be represented using *graphs*. For example, it may be possible to extract information about the *relations* between separate *objects* in the agent's field of view, thus enabling a more *causal* way of reasoning about the actions (i.e. deriving conclusions such as "performing an action on object 1 may perturb object 2, but should probably not affect object 2"). Such conclusions may greatly aid the agent's processing power on the input signal and, once again, accelerate learning. Some advances have already been made in this direction through relational networks [194], but they rely on a *complete graph of patch representations*, which uses substantially more edges than are really necessary. Making further progress reduces to the problem of *latent graph inference*, for which the current best result is the NRI model [90]—and substantial further work is required, given that the NRI's applicability is commonly limited to systems with small numbers of entities, governed by simple physical laws.

- The RL *agent* itself may internalise a graph structure with respect to itself—for example, in any kind of *robotics* or *locomotion* task, the agent's *joints* may be represented as a graph—thus allowing for a more relational approach to the output movements given to the environment. Arguably, some of the most positive results already exist for this direction, primarily in [149] and [181], where expressing the various agents' components as nodes in a graph in the MuJoCo environment [171]

allowed for more robust control behaviour.

- Lastly, I am highly interested in the problem of *continual learning*—attempting to simultaneously learn to perform *multiple tasks* within a single architecture. Prior prominent approaches to this involve *progressive neural networks* [145] and *elastic weight consolidation* [93] as well as *combining the two* [155], but I believe that this is another area where *graph-structured methodologies* can be helpful. Namely, all tasks of interest could be represented as nodes in a *graph*, with edges ideally encoding some form of information about how individual tasks are related (and therefore, how information from one may benefit the other). The graph structure could, for example, be used to govern the exchange of high-level features in a principled way. This would, significantly, allow learning signal to flow in *both* directions (from *more recently learned* tasks to earlier ones), which was a key limitation of several prior attempts. While it is at this time unclear how such a graph may be constructed, recent prominent works such as Taskonomy [195] illustrate one way in which tasks can be related for the purposes of computer vision, and could therefore serve as valuable inspiration.

Bibliography

[1] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1993–2001, 2016.

[2] Yusuf Aytar, Carl Vondrick, and Antonio Torralba. See, hear, and read: Deep aligned representations. *arXiv preprint arXiv:1706.00932*, 2017.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Scene Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

[6] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

[7] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[8] Anton L. Beer, Tina Plank, and Mark W. Greenlee. Diffusion tensor imaging shows white matter tracts between human auditory and visual cortex. *Experimental Brain Research*, 213(2):299–308, 2011.

[9] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeshwar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and Devon Hjelm. Mutual information neural estimation. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 531–540, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[10] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 7(Nov):2399–2434, 2006.

[11] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[12] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[13] Aleksandar Bojchevski and Stephan Günnemann. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *International Conference on Learning Representations*, 2018.

[14] Avishek Bose, Huan Ling, and Yanshuai Cao. Adversarial Contrastive Estimation. In *ACL*, 2018.

[15] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[16] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[17] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*, 2014.

[18] Michael Buhrmester, Tracy Kwang, and Samuel D Gosling. Amazon's Mechanical Turk: A new source of inexpensive, yet high-quality, data? *Perspectives on psychological science*, 6(1):3–5, 2011.

[19] Yaroslav Bulatov. notMNIST dataset. 2011.

[20] Dan Busbridge, Dane Sherburn, Pietro Cavallo, and Nils Y. Hammerla. Relational Graph Attention Networks, 2019.

[21] Cătălina Cangea, Petar Veličković, and Pietro Liò. XFlow: 1D-2D Cross-modal Deep Neural Networks for Audiovisual Classification. *arXiv preprint arXiv:1709.00572*, 2017.

[22] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *International Conference on Learning Representations*, 2018.

[23] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long Short-Term Memory-Networks for Machine Reading. In *EMNLP*, 2016.

[24] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *EMNLP*, 2014.

[25] François Chollet et al. Keras. `https://keras.io`, 2015.

[26] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In *International Conference on Learning Representations*, 2016.

[27] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 160–167. ACM, 2008.

[28] Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. Recurrent Batch Normalization. In *International Conference on Learning Representations*, 2017.

[29] Guillem Cucurull, Konrad Wagstyl, Arantxa Casanova, Petar Veličković, Estrid Jakobsen, Michal Drozdzal, Adriana Romero, Alan Evans, and Yoshua Bengio. Convolutional neural networks for mesh-based parcellation of the cerebral cortex. In *International Conference on Medical Imaging with Deep Learning*, 2018.

[30] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[31] Manlio De Domenico, Clara Granell, Mason A Porter, and Alex Arenas. The physics of spreading processes in multilayer networks. *Nature Physics*, 12:901 EP –, 08 2016.

[32] Andreea Deac, Petar Veličković, and Pietro Sormanni. Attentive cross-modal paratope prediction. *Journal of Computational Biology*, 2018.

[33] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[34] Misha Denil, Sergio Gómez Colmenarejo, Serkan Cabi, David Saxton, and Nando de Freitas. Programmable agents. *arXiv preprint arXiv:1706.06383*, 2017.

[35] Ming Ding, Jie Tang, and Jie Zhang. Semi-supervised Learning on Graphs with Generative Adversarial Nets. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26*, 2018.

[36] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. Learning Structural Node Embeddings via Diffusion Wavelets. In *International ACM Conference on Knowledge Discovery and Data Mining (KDD)*, volume 24, 2018.

[37] Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in neural information processing systems*, pages 1087–1098, 2017.

[38] Alberto Garcia Duran and Mathias Niepert. Learning Graph Representations with Embedding Propagation. In *Advances in Neural Information Processing Systems*, pages 5119–5130, 2017.

[39] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[40] Mark A Eckert, Nirav V Kamdar, Catherine E Chang, Christian F Beckmann, Michael D Greicius, and Vinod Menon. A cross-modal system linking primary auditory and visual cortices: Evidence from intrinsic fMRI connectivity analysis. *Human brain mapping*, 29(7):848–857, 2008.

[41] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[42] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[43] Ernesto Estrada and Jesús Gómez-Gardeñes. Communicability reveals a transition to coordinated behavior in multiplex networks. *Physical Review E*, 89(4):042819, 2014.

[44] Paolo Frasconi, Marco Gori, and Alessandro Sperduti. A general framework for adaptive processing of data structures. *IEEE transactions on Neural Networks*, 9(5):768–786, 1998.

[45] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027, 2016.

[46] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep Bayesian Active Learning with Image Data. In *Proceedings of the 34th International Conference on Machine Learning (ICML-17)*, 2017.

[47] Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*, 2016.

[48] Jonas Gehring, Michael Auli, David Grangier, and Yann N. Dauphin. A Convolutional Encoder Model for Neural Machine Translation. In *ACL*, 2016.

[49] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[50] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[51] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[52] Liyu Gong and Qiang Cheng. Adaptive Edge Features Guided Graph Attention Networks. *arXiv preprint arXiv:1809.02709*, 2018.

[53] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[54] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[55] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[56] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1319–1327, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[57] Nelson Goodman. The new riddle of induction. 1965.

[58] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *IEEE International Joint Conference on Neural Networks*, page 729–734, 2005.

[59] Clara Granell, Sergio Gómez, and Alex Arenas. Competing spreading processes on multiplex networks: awareness and epidemics. *Physical Review E*, 90(1):012808, 2014.

[60] Thomas L Griffiths, Nick Chater, Charles Kemp, Amy Perfors, and Joshua B Tenenbaum. Probabilistic models of cognition: Exploring representations and inductive biases. *Trends in cognitive sciences*, 14(8):357–364, 2010.

[61] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[62] Caglar Gulcehre, Misha Denil, Mateusz Malinowski, Ali Razavi, Razvan Pascanu, Karl Moritz Hermann, Peter Battaglia, Victor Bapst, David Raposo, Adam Santoro, and Nando de Freitas. Hyperbolic Attention Networks. In *International Conference on Learning Representations*, 2019.

[63] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304, 2010.

[64] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. *Neural Information Processing Systems (NIPS)*, 2017.

[65] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.*, 40:52–74, 2017.

[66] Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3):335–346, 1990.

[67] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[68] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[69] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[70] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[71] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[72] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with EM routing. In *International Conference on Learning Representations*, 2018.

[73] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations*, 2019.

[74] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[75] Yedid Hoshen. VAIN: Attentional multi-agent predictive modeling. In *Advances in Neural Information Processing Systems*, pages 2701–2711, 2017.

[76] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely Connected Convolutional Networks. In *CVPR*, volume 1, page 3, 2017.

[77] Drew Arad Hudson and Christopher D. Manning. Compositional Attention Networks for Machine Reasoning. In *International Conference on Learning Representations*, 2018.

[78] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456, 2015.

[79] Estrid Jakobsen, Franziskus Liem, Manousos A Klados, Seyma Bayrak, Michael Petrides, and Daniel S Margulies. Automated individual-level parcellation of Broca's region based on functional connectivity. *Neuroimage*, 2016.

[80] Simon Jégou, Michal Drozdzal, David Vázquez, Adriana Romero, and Yoshua Bengio. The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation. In *Workshop on Computer Vision in Vehicle Technology CVPRW*, 2017.

[81] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th International Conference on the World Wide Web*, pages 271–279. ACM, 2003.

[82] Michael I Jordan. Serial order: A parallel distributed processing approach. In *Advances in psychology*, volume 121, pages 471–495. Elsevier, 1997.

[83] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*, pages 1–12. IEEE, 2017.

[84] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350, 2015.

[85] Lukasz Kaiser, Aidan N Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. One model to learn them all. *arXiv preprint arXiv:1706.05137*, 2017.

[86] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

[87] Laurynas Karazija, Petar Veličković, and Pietro Liò. Automatic Inference of Cross-Modal Connection Topologies for X-CNNs. In *International Symposium on Neural Networks*, pages 54–63. Springer, 2018.

[88] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[89] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

[90] Thomas N. Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard S. Zemel. Neural Relational Inference for Interacting Systems. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 2693–2702, 2018.

[91] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[92] Thomas N Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*, 2017.

[93] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, page 201611835, 2017.

[94] Mikko Kivelä, Alex Arenas, Marc Barthelemy, James P Gleeson, Yamir Moreno, and Mason A Porter. Multilayer networks. *Journal of complex networks*, 2(3):203–271, 2014.

[95] C Kleiser, N Wawro, M Stelmach-Mardas, H Boeing, K Gedrich, H Himmerich, and J Linseisen. Are sleep duration, midpoint of sleep and sleep quality associated with dietary intake among Bavarian adults? *European journal of clinical nutrition*, 71(5):631, 2017.

[96] Wouter Kool, Herke van Hoof, and Max Welling. Attention, Learn to Solve Routing Problems! In *International Conference on Learning Representations*, 2019.

[97] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[98] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[99] Neal Lathia, Veljko Pejovic, Kiran K Rachuri, Cecilia Mascolo, Mirco Musolesi, and Peter J Rentfrow. Smartphones for large-scale behavior change interventions. *IEEE Pervasive Computing*, (3):66–73, 2013.

[100] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[101] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[102] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[103] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[104] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[105] John Boaz Lee, Ryan A Rossi, Sungchul Kim, Nesreen K Ahmed, and Eunyee Koh. Attention Models in Graphs: A Survey. *arXiv preprint arXiv:1807.07984*, 2018.

[106] Anming Li, Hareton Leung, and Yvette Lui. Friend recommendation for weight loss app. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pages 1257–1264. ACM, 2014.

[107] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *International Conference on Learning Representations*, 2016.

[108] Edgar Liberis, Petar Velickovic, Pietro Sormanni, Michele Vendruscolo, and Pietro Liò. Parapred: Antibody Paratope Prediction using Convolutional and Recurrent Neural Networks. *Bioinformatics*, 1:7, 2018.

[109] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive Sentence Embedding. In *International Conference on Learning Representations*, 2017.

[110] Qing Lu and Lise Getoor. Link-based classification. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 496–503, 2003.

[111] Edith Talina Luhanga, Akpa Akpro Elder Hippocrate, Hirohiko Suwa, Yutaka Arakawa, and Keiichi Yasumoto. Towards proactive food diaries: a participatory design study. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 1263–1266. ACM, 2016.

[112] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, page 3, 2013.

[113] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[114] Brian W Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.

[115] James L McClelland. The interaction of nature and nurture in development: A parallel distributed processing perspective. *International perspectives on psychological science*, 1:57–88, 1994.

[116] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[117] Dmytro Mishkin and Jiri Matas. All you need is a good init. In *International Conference on Learning Representations*, 2016.

[118] Tom M Mitchell. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ. New Jersey, 1980.

[119] Andriy Mnih and Koray Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in neural information processing systems*, pages 2265–2273, 2013.

[120] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[121] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proc. CVPR*, volume 1, page 3, 2017.

[122] Arlet V Nedeltcheva, Jennifer M Kilkus, Jacqueline Imperial, Kristen Kasza, Dale A Schoeller, and Plamen D Penev. Sleep curtailment is accompanied by increased intake of calories from snacks. *The American journal of clinical nutrition*, 89(1):126–133, 2008.

[123] Allen Newell. Physical symbol systems. *Cognitive science*, 4(2):135–183, 1980.

[124] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011.

[125] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning Convolutional Neural Networks for Graphs. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 2014–2023, 2016.

[126] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems*, pages 271–279, 2016.

[127] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, page 125, 2016.

[128] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation Learning with Contrastive Predictive Coding. *arXiv preprint arXiv:1807.03748*, 2018.

[129] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[130] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.

[131] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[132] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[133] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. DeepInf: Social Influence Prediction with Deep Learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2110–2119. ACM, 2018.

[134] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. Efficient Learning of Sparse Representations with an Energy-based Model. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, pages 1137–1144, Cambridge, MA, USA, 2006. MIT Press.

[135] Scott Reed and Nando De Freitas. Neural Programmer-Interpreters. In *International Conference on Learning Representations*, 2016.

[136] Jimmy SJ Ren, Yongtao Hu, Yu-Wing Tai, Chuan Wang, Li Xu, Wenxiu Sun, and Qiong Yan. Look, Listen and Learn-A Multimodal LSTM for Speaker Identification. In *AAAI*, pages 3581–3587, 2016.

[137] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 385–394. ACM, 2017.

[138] Daniel Ritchie, Paul Horsfall, and Noah D Goodman. Deep amortized inference for probabilistic programs. *arXiv preprint arXiv:1610.05735*, 2016.

[139] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. FitNets: Hints for Thin Deep Nets. In *International Conference on Learning Representations*, 2015.

[140] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[141] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[142] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, 2000.

[143] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.

[144] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[145] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[146] Andrei A. Rusu, Matej Večerík, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-Real Robot Learning from Pixels with Progressive Nets. In *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 262–270. PMLR, 13–15 Nov 2017.

[147] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3856–3866, 2017.

[148] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016.

[149] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin A. Riedmiller, Raia Hadsell, and Peter Battaglia. Graph Networks as Learnable Physics Engines for Inference and Control. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 4467–4476, 2018.

[150] Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. Relational recurrent neural networks. In *Advances in neural information processing systems*, pages 7310–7321, 2018.

[151] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Tim Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4967–4976, 2017.

[152] Natsuko Sato-Mito, Satoshi Sasaki, Kentaro Murakami, Hitomi Okubo, Yoshiko Takahashi, Shigenobu Shibata, Kazuhiko Yamada, Kazuto Sato, Freshmen in Dietetic Courses Study II group, et al. The midpoint of sleep is associated with di-

etary intake and dietary behavior among young Japanese women. *Sleep medicine*, 12(3):289–294, 2011.

[153] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations*, 2014.

[154] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[155] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & Compress: A scalable framework for continual learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4528–4537, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[156] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

[157] Chao Shang, Qinqing Liu, Ko-Shin Chen, Jiangwen Sun, Jin Lu, Jinfeng Yi, and Jinbo Bi. Edge Attention-based Multi-Relational Graph Convolutional Networks. *arXiv preprint arXiv:1802.04944v1*, 2018.

[158] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

[159] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[160] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[161] Alessandro Sperduti and Antonina Starita. Supervised Neural Networks for the Classification of Structures. *Trans. Neur. Netw.*, 8(3):714–735, May 1997.

[162] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[163] Nitish Srivastava and Ruslan R Salakhutdinov. Multimodal learning with deep boltzmann machines. In *Advances in neural information processing systems*, pages 2222–2230, 2012.

[164] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

[165] Aravind Subramanian, Pablo Tamayo, Vamsi K Mootha, Sayan Mukherjee, Benjamin L Ebert, Michael A Gillette, Amanda Paulovich, Scott L Pomeroy, Todd R Golub, Eric S Lander, et al. Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences*, 102(43):15545–15550, 2005.

[166] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[167] Jan Svoboda, Jonathan Masci, Federico Monti, Michael Bronstein, and Leonidas Guibas. PeerNets: Exploiting Peer Wisdom Against Adversarial Attacks. In *International Conference on Learning Representations*, 2019.

[168] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[169] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[170] Tijmen Tieleman and Geoffrey Hinton. 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, 2012.

[171] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[172] Gregor Urban, Krzysztof J Geras, Samira Ebrahimi Kahou, Ozlem Aslan, Shengjie Wang, Rich Caruana, Abdelrahman Mohamed, Matthai Philipose, and Matt Richardson. Do deep convolutional nets really need to be deep and convolutional? In *International Conference on Learning Representations*, 2017.

[173] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[174] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[175] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep Graph Infomax. In *International Conference on Learning Representations*, 2019.

[176] Petar Veličković, Duo Wang, Nicholas D Lane, and Pietro Liò. X-CNN: Cross-modal convolutional neural networks for sparse datasets. In *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, pages 1–8. IEEE, 2016.

[177] Petar Veličković, Laurynas Karazija, Nicholas D. Lane, Sourav Bhattacharya, Edgar Liberis, Pietro Liò, Angela Chieh, Otmane Bellahsen, and Matthieu Vegreville. Cross-modal Recurrent Models for Weight Objective Prediction from Multimodal Time-series Data. In *Proceedings of the 12th EAI International Conference on Pervasive Computing Technologies for Healthcare*, PervasiveHealth '18, pages 178–186, New York, NY, USA, 2018. ACM.

[178] Petar Veličković and Pietro Liò. Molecular multiplex network inference using Gaussian mixture hidden Markov models. *Journal of Complex Networks*, 2015.

[179] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order Matters: Sequence to sequence for sets. In *International Conference on Learning Representations*, 2016.

[180] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM, 2016.

[181] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018.

[182] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community Preserving Network Embedding. In *AAAI*, pages 203–209, 2017.

[183] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph CNN for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018.

[184] Sinead Watson, Jayne V Woodside, Lisa J Ware, Steven J Hunter, Alanna McGrath, Christopher R Cardwell, Katherine M Appleton, Ian S Young, and Michelle C McKinley. Effect of a web-based behavior change program on weight loss and cardiovascular risk factors in overweight and obese adults at high risk of developing cardiovascular disease: Randomized controlled trial. *Journal of medical Internet research*, 17(7), 2015.

[185] Boris Weisfeiler and AA Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

[186] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory Networks. In *International Conference on Learning Representations*, 2015.

[187] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

[188] Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Josh Tenenbaum. Learning to see physics via visual de-animation. In *Advances in Neural Information Processing Systems*, pages 153–164, 2017.

[189] Weiping Yang, Jingjing Yang, Yulin Gao, Xiaoyu Tang, Yanna Ren, Satoshi Takahashi, and Jinglong Wu. Effects of Sound Frequency on Audiovisual Integration: An Event-Related Potential Study. *PLoS ONE*, 10(9):1–15, 09 2015.

[190] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting Semi-Supervised Learning with Graph Embeddings. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 40–48, New York, New York, USA, 20–22 Jun 2016. PMLR.

[191] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *Proceedings of the 24th ACM SIGKDD International Con-*

ference on Knowledge Discovery & Data Mining, KDD '18, pages 974–983, New York, NY, USA, 2018. ACM.

[192] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems*, pages 4805–4815, 2018.

[193] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.

[194] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter Battaglia. Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations*, 2019.

[195] Amir R Zamir, Alexander Sax, William Shen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling Task Transfer Learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3712–3722, 2018.

[196] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[197] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. GaAN: Gated Attention Networks for Learning on Large and Spatiotemporal Graphs. In *Uncertainty in Artificial Intelligence (UAI)*, pages 339–349, 2018.

[198] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pages 912–919, 2003.

[199] Peiye Zhuang, Alexander G Schwing, and Oluwasanmi Koyejo. Hallucinating brains with artificial brains. 2018.

[200] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017.