Universitat Politècnica de Catalunya
Department of Signal Theory and Communication

Music Generation with Deep Learning Techniques

Transformer-based Generative Adversarial Network

Author: Pau Lozano
Supervisor: Philippe Salembier

October 7th, 2022

Grau en Ciència i Enginyeria de Dades

Facultat d'Informàtica de Barcelona - FIB
Escola Tècnica Superior d'Enginyeria de Telecomunicacions de Barcelona - ETSETB
Facultat de Matemàtiques i Estadística - FME

Abstract

In this bachelor's thesis, we propose a deep learning model for generating music. Our model is based on the Transformers architecture and is pre-trained with the MaestroV2 dataset. We fine-tune our generative model using a Generative Adversarial Network, with the Gumbel-Softmax technique to allow backpropagation. Our results show that the discriminator of the GAN creates a bottleneck, and that the results are promising but do not assess that the fine-tuned generator model outperforms the original one.

En aquesta tesis, presentem un model de deep learning per generar música. El nostre model es basa en l'arquitectura Transformer i es pre-entrena a partir de la base de dades MaestroV2. Apliquem fine-tuning al model generador utilitzant una Generative Adversarial Network, utilitzant la tècnica de la Gumbel-Softmax per permetre la retropropagació a través del model. Els resultats indiquen que el discriminador de la GAN genera un coll d'ampolla, i que els resultats, tot i que són prometedors, no són suficients per clarificar si el model optimitzat per la GAN té un millor rendiment que l'original.

En esta tesis presentamos un modelo de deep learning para generar música. Nuestro modelo se basa en la arquitectura Transformer y se pre-entrena con la base de datos MaestroV2. Aplicamos fine-tuning al modelo generador utilizando una Generative Adversarial Network, utilizando la técnica de la Gumbel-Softmax para permitir la retropropagación a través del modelo. Los resultados indican que el discriminador de la GAN genera un cuello de botella, y que los resultados, pese a ser prometedores, no permiten clarificar si el modelo optimizado por la GAN muestra un mejor rendimiento que el original.

1. Introduction and State of the Art

1.1. Introduction

Among all the existing artistic disciplines, music is clearly one of the most enjoyed and practiced by humans. Music is one of the oldest forms of human expression, with roots tracing back to before the dawn of civilization. It is a universal language that has the power to transcend cultural barriers and bring people together. Nevertheless, musicians throughout history have pushed the boundaries of what music represented, how it sounded and particularly how it was interpreted, instruments have evolved, knowledge on acoustics did as well, and with the arrival of computers and software, it also has completely changed the way we consume it. New technologies have democratized music to an unprecedented level and nowadays we can enjoy all our favorite artists in the palm of our hand.

These advancements in technology do also include the rapid improvements Artificial Intelligence is experiencing in many fields and use-cases. The presence of Artificial Intelligence (AI) in the arts domain is not an exception and has grown considerably in recent years. In fields such as painting, AI is becoming extremely popular for the well-proven capabilities of deep learning models for text-to-image generations (**Dall•E, Stable Diffusion etc**), image correction and others.
In the music field, AI is now providing new ways to experience music, as well as creating it. AI music has been around for a while now, with the first fully-composed AI song released in 1957 (**Illiac Suite for String Quartet)**. However, it is only in recent years with the advances made in the deep learning models and the increase of computational power that A.I. has begun to be used in a more creative way, with A.I. composed songs being released on major streaming platforms such as Spotify (**SKYGGE**).

1.2. Existing approaches

At the time of writing this thesis, many approaches have been proposed with varying degrees of success for the Music Generation challenge. Investigating and covering them all would require another entire thesis [1] so in this memory only the most popular and representative ones will be discussed. In this section, we will briefly review the different data formats

that can be used to process music and then some model architectures that are being applied fed by this data.

Data Formats

There is a strong relationship between music and other human languages: they are meant to share some kind of information, they do so by encoding it in a temporal sequence, and they require dependencies between the different elements in that sequence to form comprehensive and coherent messages.
It is then clear to see that music representations and text face and solve similar challenges to accomplish a proper transcription of their language.

Music can be represented in many different ways, some are easier to understand than others and that applies to computers as well.
The preferred way to represent music has historically been the score. A score is a document containing musical notation (musical language) and is a literal transcription of a musical piece. It usually represents the notes with different symbols (eights, quarters, halves and wholes), rests (of varying lengths) and many other symbols that represent characteristics like the time signatures, clefs, key signatures, tempo and many others.
Scores can represent music with great precision and are easily readable for humans with previous formation.

There are some differences between scores and other language representations such as text.

Text is our primary tool of communication. Text transcriptions are designed to share information in an effective and efficient way. For that reason, they usually have a big alphabet (26 letters are used in English, 27 in Spanish and 26 in Catalan) but do not have a wide range of symbols providing information about the form of the message (some exceptions would be the symbols ! and ?, that actively modify the meaning of the text or the way it has to be read).

Scores, on the other hand, have a limited alphabet -there are only 7 different notes- but a wide variety of symbols that specify the way those notes have to be played. “Music messages” are more susceptible to meaning and information variations if not played in a specific way. E.g: While a text will always mean the same thing whether the reader is faster

or slower, music falls down into pieces if the tempo is not correctly followed.

Because of these differences, the way we encode information in a computer needs to be rethinked, and the biggest exponent in musical data processing is MIDI. Musical Instrument Digital Interface (MIDI) is a technical standard that describes a communications protocol, digital interface, and electrical connectors that connect a wide variety of electronic musical instruments, computers, and related audio devices for playing, editing, and recording music. It does so by storing the information about the music into events.

MIDI is arguably the most popular data format when it comes to processing music with a computer. Nevertheless, the way this data is used and the amount of information taken into account can vary.

Piano-roll

A piano-roll is a binary matrix representing notes being played during T time steps [2]. It is a very understandable data representation that follows a simple assumption: each time step has the same length (whether it is in seconds or milliseconds).

The horizontal axis usually represents the different pitches (commonly 128, which is what MIDI files can support) and the vertical axis defines the temporal dimension (image 1).

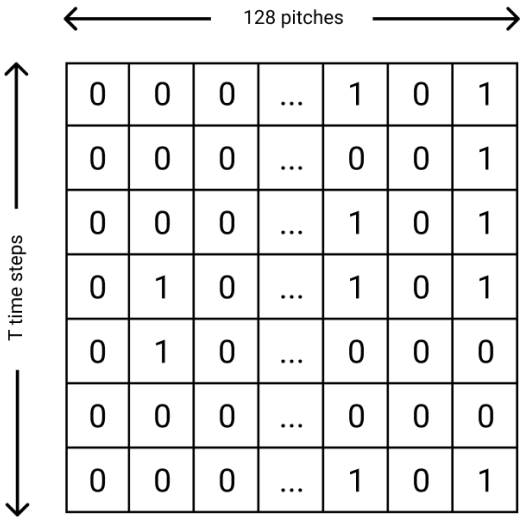


Image 1. An example of a piano-roll matrix

This approach has some benefits and problems. On one hand, being a binary matrix makes this data format very manageable and understandable both by humans and computers. It accurately represents the notes and can even be modified to express the volume of those notes by replacing the 1s in the matrix with other values (usually from 1 to 100).
The limitations on the piano roll are mainly on the degree of precision they can achieve in the temporal domain. While this representation allows very small time-steps, these will inevitably increase the number of rows that the matrix has. In other words, trying to be very precise with the temporal

dimension increases the size of the data to be processed. For example, for the same amount of time, say 1 second, using time steps of 5ms instead of 20ms quadruples the amount of data that the model needs to process, making it much slower and also suffering from memory loss.
On top of that, being very precise with the time dimension also causes the matrix to have a lot of row repetitions (if a single note is being played during 1 second, the piano roll will look the same for hundreds of rows) which is not efficient at all. Finally, piano rolls also suffer from data sparsity. For each time step only a few notes amongst the 128 columns are being played, which makes the model process a lot of data that is redundant and potentially unnecessary.

Events

A different approach for the data format problem is mapping information about the music into events [3]. The main idea here is that any single action is stored individually, which converts the music into a sequence of events with a specific order. As MIDI already provides a standard event-based format, it is common to take advantage of it. Nevertheless, while MIDI implements a wide variety of events, these are not specifically designed to feed a Machine Learning model, so it is common to use parsing libraries that reduce the types of events to just a few. It depends on the library, but usually these events are Note on, Note off, Absolute time and Volume.

Using the same example as in the previous section, playing one note during 1 second only requires 4 events:
Volume, Note on, Time shift of 1 second, Note off.
In comparison, a piano-roll would need hundreds of rows (of a few milliseconds each) of binary data to replicate the same.

This approach however also has some limitations, especially in the temporal dimension. It is not an option to encode any possible time shift into an event, that would turn out to be thousands of possible events, so usually the time shifts are stored into 100 to 200 different events that approximate the real time-shift.

Raw audio

Using raw audio [4] is the last of the most common data formats and it completely changes the scope of the problem, mainly because it is not an accurate representation of a musical piece but instead an accurate representation of a specific musical performance. Using raw audio no longer presents temporal limitations of any kind and has a richer palette of potential musical representations, but it comes with an extra cost in computation and the resources needed to train a model.

Deep Learning Architectures

As has already been discussed at the beginning of the section, music can easily be understood as a language. It shares information encoded in a sequence along the temporal dimension, so it makes sense to use architectures that have already been successful with Natural Language.

Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM)

This is one of the most naive approaches one can come up with, especially after some formation on deep learning. RNN generative models are well known for their decent memory capacities, which makes them an interesting option for music generation. RNNs [5] together with Long Short-Term Memory [6] architectures have been mainly used in NLP for text generation/continuation and until the arrival of Transformers [7] were considered the state-of-the-art, although now their utilization has generally been relegated to simpler tasks.

During my last university year I did an “introduction to research” course in which I tried this approach: I used a double-layered LSTM architecture as a generative model whose output were single steps of a piano roll (images 2 and 3). Although the model was decent at preserving the clef, the melodies and tempo structures were on average quite poor, concluding that while LSTMs could potentially “understand” and correctly generalize some music theory concepts like the scale, using more advanced methods was recommended.

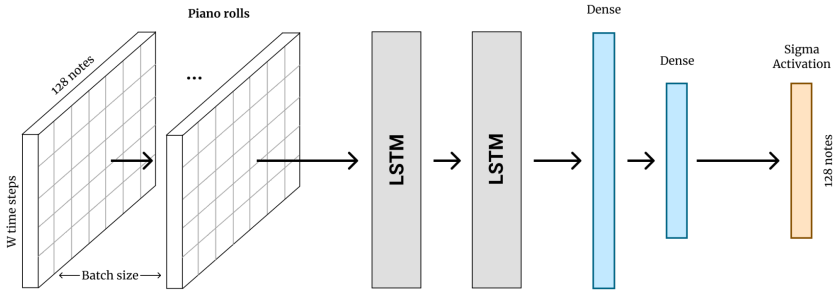


Image 2. Architecture of the model proposed during the course "Introduction to research"

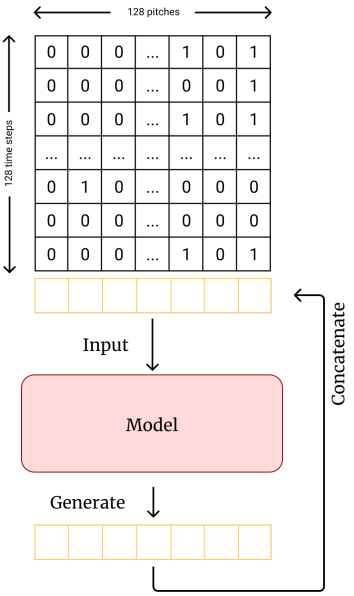


Image 3. A single generation step of the model proposed

Transformers

Transformers [7] are considered the state-of-the-art for many use-cases. On demanding NLP tasks they have been the undisputable choice during the last years for their better performance when compared to LSTMs and RNNs, and have empowered the most capable models we have ever had in deep learning like GPT-3 [8]. Transformers are an attention-dependent architecture, and were first introduced in the context of machine translation in order to allow parallel computation and reduce drops in performance due to long dependencies.
Using the NLP domain as an example, Transformers can be summarized in the following key points:

- Non sequential. Sentences are processed as a whole rather than word by word.
- Self-Attention. Used to compute similarity scores between words in a sentence.
- Positional embeddings. Introduced to replace recurrence. The idea is to use fixed or learned weights which encode information related to a specific position of a token in a sentence.

The first being the main reason for the much better long-term memory Transformers have compared with LSTMs and RNNs.

In the Music Generation domain, transformers and similar variants have been used by many with satisfactory results. Examples of these are Tensorflow's Magenta Music Transformer [9] or OpenAI's Musenet [10], the last one using a very similar architecture than GPT-2. [11]

Non time-iterative approaches: Generative Adversarial Networks (GAN)

Generative Adversarial Networks [12] have been until very recently the most popular generative models in the image domain. Even though Diffusion models [13] seem to be the trendiest architectures lately, GANs have been consistently delivering with outstanding performance for some years already in a very diverse spectrum of use-cases like image translation, image generation and others [14].
For music, GANs also have their place within the literature with interesting results. In MuseGAN [15], a very popular experiment, the piano roll has been understood not like a sequence of notes but like a score that can be generated in a non temporal-recursive manner, meaning that the whole sequence is generated all from random noise and mapped to the final score over the course of many iterations (image 4).

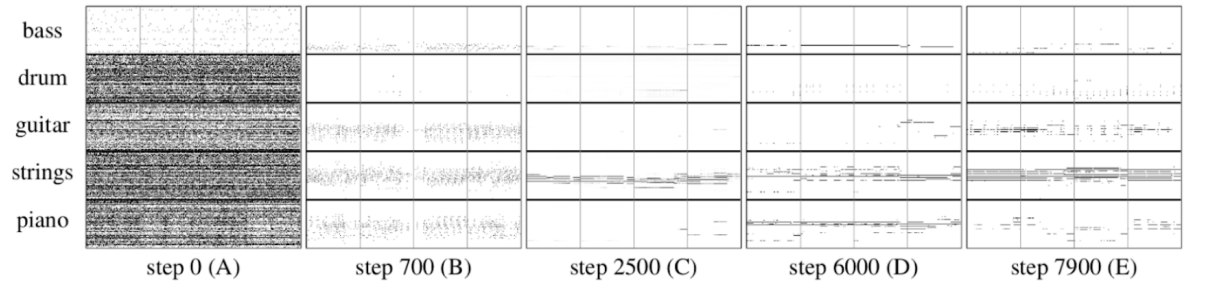


Image 4. Evolution of the generated pianorolls as a function of update steps

2. Discussion, intuitions and assumptions

In this section the assumptions and intuitions behind our solution proposition will be discussed before presenting it. We will also provide background and justify some of the choices made during the realization of the project.

2.1. What kind of music do we want to make?

This is an important decision. Any musical genre has its own peculiarities, and the creative challenge can fall in different facets depending on it. While classical music usually displays a very wide range of musical resources and offers complex and detailed melodies, other genres like pop, rock, hip hop etc take a smaller portion of those resources but use them in different ways that might be just as hard to master.

However, classical music is the most common genre in the literature of music generation. The most popular datasets also follow this trend, and using them has become a “benchmark” on itself. For this reason, we will be training a “classical music” generator as well.

2.2. What makes music good or bad?

Defining metrics to evaluate the goodness of the results is crucial in any research project. The goals are usually set to achieve the best as possible in those metrics, which provides a scope to observe the results and often justifies the approach and decisions taken.

However, finding a good metric is not always an easy task. In generation models, such as text generators, image generators or in our case, music generators, deciding what makes the outputs good or bad can become extraordinarily difficult and hard to calculate numerically. For the case of text generation, there are many approaches that have been proposed, like using the perplexity of the generations, BLEU [16] or Rouge [17].
For artistic tasks, it only gets worse. Not only does the model have to produce results similar to the reference (like BLEU and Rouge calculate) but it also has to correctly abstract the “artistic” facet of the data, which is in itself a very subjective thing to tell.

For that reason, and because there does not seem to exist a consensus on what metrics are the best for music evaluation, we propose to slightly change the scope of the problem: We will aim to make our generations “sound” as humane as possible. By doing so we are trying our generator not to produce the best musical pieces as possible (because there is no metric proposed so far that can possibly evaluate that) but instead to produce music that a human evaluator may misclassify as an actual human creation, whether if the evaluator considers the music to be good or bad.

2.3. Assumptions and idea propositions

According to this new scope of evaluation discussed in the previous section, during the planification of the project, we aimed to go a step further than the current state-of-the-art and assumed the following statement:

An already trained music generation model could be improved by making its generations more “humane”.

A few ideas were considered and discarded due to the limitations they carried:

- **Using different data sets** to fine-tune the model and ensure there is no bias and the model can properly generalize. E.g: Using many genres of music might force the model to learn more abstract patterns present in all kinds of music.
  However, this method relies on the amount, the quality and the format of the available data, a limitation that sometimes can not be overcomed.

- **Using reinforcement learning** to fine-tune the model. To use reinforcement learning a reward metric is usually implemented [18], which is not trivial at all for music generation. We imagined a training process based on human evaluation, the model would need to convince the human that the piece was not artificially generated, and to do so maybe find new ways to make music more “humane”. However, the limiting factor here is the cost this approach implies. Unlike recommender systems and other cases where reinforcement learning can be applied using human evaluation [19], in this case

evaluating complete or fragmented generations would require an excessive amount of time for each training step. The training process would be extremely tedious and would still be biased towards the musical taste of the evaluators.

There is a third idea that we came up with, which had potential and we found worth investigating:

- **Using a discriminator model** specifically trained to distinguish between original and artificial music from the generator and use it like in a GAN. The generator and discriminator would be connected and the generator would try to fool the discriminator as often as possible. Both the generator and discriminator would be fine-tuned during the training process to not lose against each other.

This last approach did not have any of the previous limitations:

- The discriminator can be specifically built for any data format.
- After pre-training the generator, no data is required.
- No human evaluation is required.

We opted to proceed with this last idea, aware that it could also have some limitations. Particularly, as we would not have complete control on what changes the generator suffered, it may find a way to fool the discriminator that did not necessarily make the musical piece look more human. However, optimizing hyper-parameters like the learning rate and saving checkpoints of the models every epoch should allow us to prevent that from happening.

2.4. What generator to choose

As seen in the introduction and previous sections, many approaches have been proposed with satisfactory results for the generative model. That is especially true for the Transformers idea, with cases like OpenAI's MuseNet [10] being particularly well-received and with its results being a step ahead of the rest of experiments or products proposed so far.

For that reason, using Transformers seems like a good idea. First, they are great at managing long time-structured sequences, which is a crucial requirement in music given that musical pieces usually condense a lot of complexity during long periods of time. Second, they do not work

iteratively, meaning that the training time is usually faster by applying parallel computing (LSTMs and RNNs need the previous step prediction to move forward, which is intrinsically non-parallelizable). Third, Transformers rely completely on “attention”, a method to calculate the relevance of a past state to better predict the outcome of the present. This attention system is extremely important to guarantee songs will remain consistent throughout all the generated parts in many aspects like the melody, the scale, tempo and others.

While choosing to use Transformers does not guarantee that the generator model will be necessarily good on its generations, it is true that we are taking the best approach discovered so far.

2.5. What discriminator to choose?

The discriminator faces a basic yet complex task trying to classify between artificial and original musical pieces. Text classification is a well-studied problem and the insights from that domain can be applied for the music one. BERT, XLNet, and RoBERTa are a few of the most popular and powerful models that have been successfully applied for text classification and all are based in the Transformer architecture.

The same benefits that Transformers offer, already discussed above, for the generation problem can be applied for the classification task. For that reason, we will opt to implement a transformer architecture as well in our discriminator.

3. Goals of the project

After reviewing the intuitions and assumptions, we define the objectives of this project according to the ideas presented in the previous section.

The objectives of the project are as follows:

1. Defining a Generative Adversarial Network (GAN) to discern between artificial and human music compositions.

2. To use this GAN to fine-tune the generative model.

3. To empirically assess if the fine-tuned generative model outperforms the original one.

In the next sections, the solution proposed to achieve these goals, the experiments performed and the final results will be explained.

4. Proposed Solution

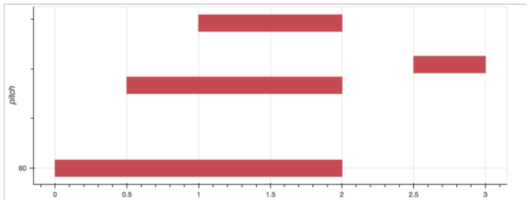
We now present the architecture of the model as we have built it and tested it.

4.1. Data format and dataset

We are using the MaestroV2 dataset [20], a MIDI collection of more than 172 hours of virtuosic piano performances captured with fine alignment (~3 ms) between note labels and audio waveforms. The MIDI files are then processed and converted into sequences of events, which is what the model will receive as inputs. This MIDI processor is provided by Ian Simon and Sageev Oore [21] and assembles the sequences using the following kinds of events:

- **128 note-on events**, one for each of the 128 MIDI pitches. These events start a new note.
- **128 note-off events**, one for each of the 128 MIDI pitches. These events release a note.
- **100 time-shift events** in increments of 10 ms up to 1 second. These events move forward in time to the next note event.
- **32 velocity events**, corresponding to MIDI velocities quantized into 32 bins. These events change the velocity applied to subsequent notes. It is easier to understand this velocity as "volume", given that the faster a piano key is played, the louder it sounds.

Then, the model will receive sequences of varying sizes and combinations of all the 388 possible events. Image 5 is used from the work of Oore et al.



```
SET_VELOCITY<80>, NOTE_ON<60>
TIME_SHIFT<500>, NOTE_ON<64>
TIME_SHIFT<500>, NOTE_ON<67>
TIME_SHIFT<1000>, NOTE_OFF<60>, NOTE_OFF<64>,
NOTE_OFF<67>
TIME_SHIFT<500>, SET_VELOCITY<100>, NOTE_ON<65>
TIME_SHIFT<500>, NOTE_OFF<65>
```

Image 5. Example of the data format. It can be read as follows: velocity 80 is set, note 60 starts being played, 500ms pass, note 64 is played, 500ms pass, note 60 starts being played, 500ms pass, all the active notes (60,64 and 67) stop being played. 500ms pass, velocity is set to 100, note 65 starts being played, 500ms pass, note 65 stops playing.

4.2. Generator

As already introduced, the generator model (Image 6) is a multi-layer Transformer architecture. It is inspired by the model proposed by Huang,

C.-Z.A. *et al.* [9] with variations in the sampling process that will be explained in 4.5. These are the main characteristics:

- 6 Transformer layers with 8 attention heads each and internal representation size of 512.
- Embeddings of size 512.
- Positional encoding.
- Maximum sequence size of 2048.
- A dropout of 0.1 is set for a better generalization.

A fully connected layer of dimension 1024 is added at the output of the Transformer stack before applying a Softmax activation function.

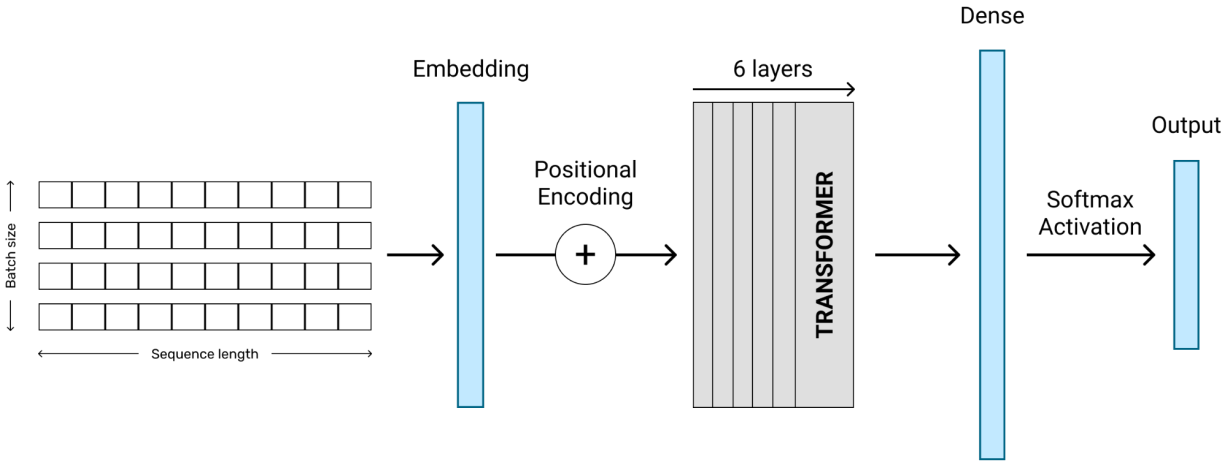


Image 6. Diagram of the Generator Architecture

The model reads a sequence of events (as already explained, the size of the vocabulary is 388) and returns a vector of probabilities of size 388, a distribution, containing the probabilities of each event to be the next in the sequence. For a long generation, we sample from this distribution and

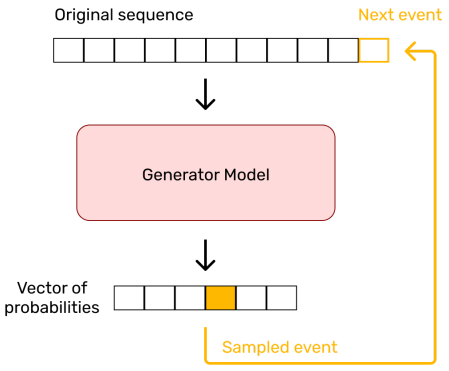


Image 7. Diagram of a generation step

append the picked event to the original sequence (Image 7). We repeat this process as long as we want until we get a full musical piece.

Prior to fine-tuning this model with the GAN, we pre-train it using the MaestroV2 dataset and the Cross Entropy as the loss function.

4.3. Discriminator

The architecture of the discriminator (Image 8) remains almost the same as in the generator model, trying to balance as much as possible the capabilities of both adversaries. The biggest modifications are on the last layers, we add a new fully-connected layer with size of 1024 before applying the activation function, which now is the Sigmoid.

Both the generator and discriminator have around 14.000.000 parameters each.

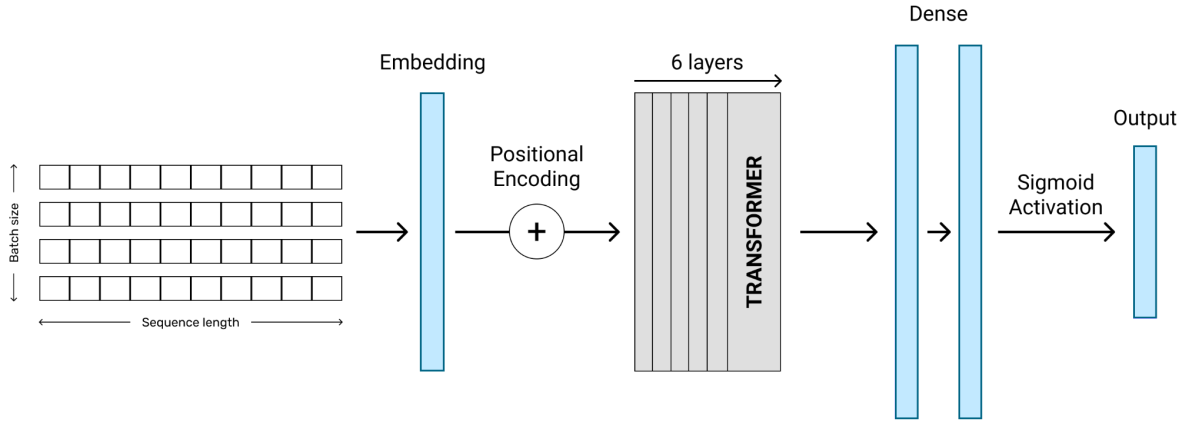


Image 8. Diagram of the Discriminator Architecture

While the output of the generator model is a vector of size 388 indicating the probability of each event to be the next one in the sequence, the output of the discriminator is a vector of size *batch size* indicating the probabilities of the input songs to be real or artificial (Image 9).

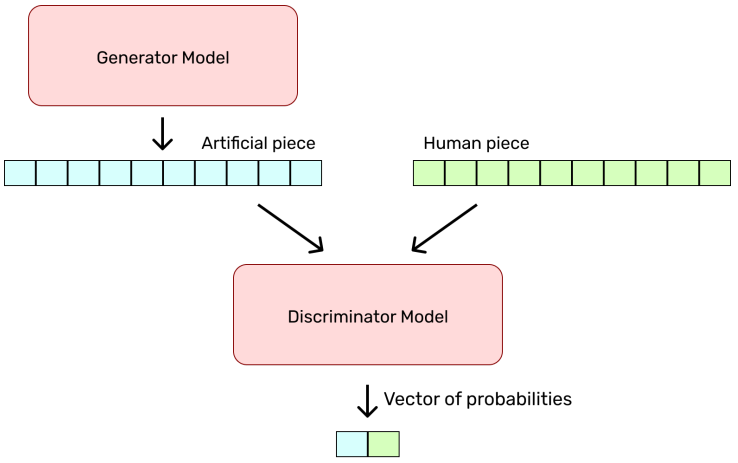


Image 9. Diagram of a discrimination step of an artificial and an origianl piece

The discriminator is pre-trained with a few samples of original and artificial songs so it does not start the GAN training with a disadvantage.

4.4. Transformed-based Generative Adversarial Network

Our proposition to improve the generator without human evaluation consists in connecting the Generator with the Discriminator in both directions and closing the model cycle. The discriminator has to force the generator to slightly modify its creations to the point where original and generated data have no visible differences from the discriminator perspective.

Regarding the implementation, on one side we have the generator, pre-trained with the MaestroV2 dataset, and on the other side, we have the discriminator that listens to both original and artificial sequences and predicts which one is made by a human.

The model has been trained as follows:

1. A random sequence of events with size of 384 is taken from the original pieces dataset. From this sequence we split in two:
    a. The first 192 events will be used as a seed for the generator.
    b. The last 192 events will be seen as the target.

2. The first half of the sequence is passed to the generator, who starts predicting events until the sequence reaches a length of 384 (Image 10).

Image 10. The original sequence of size 384 is splitted. The first half is used as the seed for the generator, the second one is used as the target

3. The generated sequence is passed through the discriminator, who predicts if it is original or artificial, and we calculate the loss. As this is a binary classification problem, we use the Binary Cross Entropy

(BCE) loss function, and as it is common in GANs, we label the generated piece as if it actually was original, given that we want to obtain a small loss if the generator succeeds at making the discriminator think the piece to be original (Image 11). The loss value is then back-propagated to the Generator model.

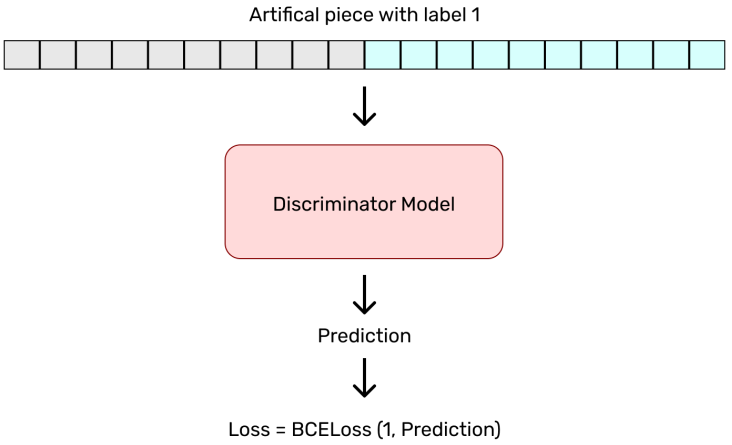


Image 11. Generator loss is calculated between the prediction of the model and the label 1.

4. We focus now on updating the discriminator. We start by passing the original sequence through the discriminator and calculate the Loss using again the “original” label. Then, we take the generated sequence and its prediction calculated in the previous step and calculate the Loss again, but this time, as the piece is artificial and we want to encourage the discriminator to reject it, we use the “artificial” label. We combine the two losses by averaging them and back-propagate the final loss to the discriminator. (Image 12)

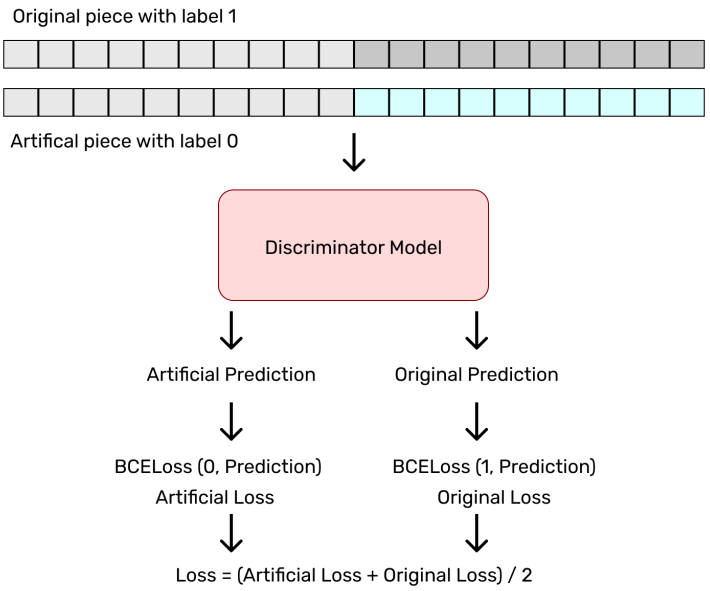


Image 12. Total discriminator loss is calculated averaging the artificial and original losses.

5. Repeating the steps 1, 2, 3, 4 trains the model fine-tuning both the generator and the discriminator (Image 13).

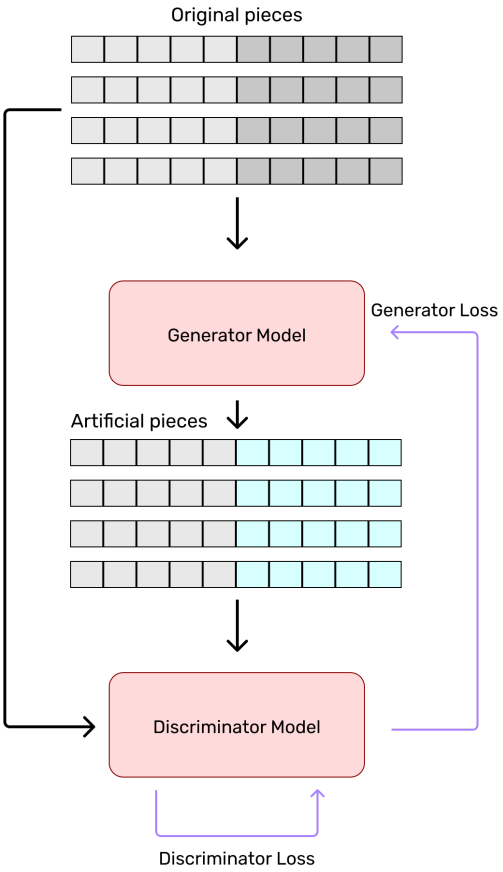


Image 13. Diagram of the model during training.

4.5. The Gumbel Softmax

An observing reader might have caught that there is an issue to address before training the model. We are backpropagating through the generator after a full sequence generation, which means that every time the generator predicts the following event it is actually sampling from a categorical distribution. However, the sampling process from a categorical distribution is **not differentiable**, meaning that backpropagation will not work. In other words, every time the generator is adding a new event to the sequence it is performing a non-differentiable operation, meaning that the gradient can not be propagated backwards and that we can not update the model with the loss obtained from the discriminator model. So we need to find another way to generate sequences of events that will allow us to perform backpropagation.

This problem has been faced before in other research fields like NLP and a few solutions have been proposed [27]. However, we wanted a solution that could be implemented without re-thinking the generator model. For this reason, we used the **Gumbel Softmax** [23, 25] after seeing how it had been successfully applied for GANs generating text [28].

Explaining the whole mathematical principle would require at least 5 pages in this memory and it would not add any deeper perspective into the scope of this project, so we will briefly summarize what the Gumbel Softmax achieves and what it allows us to do. We strongly recommend reading Emma Benjaminson's blog about the topic [22] or the articles that presented this approach from Jang et al. [23, 24] for a better understanding of the matter. We take references from Emma's blog in this section.

The Gumbel Softmax uses the reparametrization trick to convert a sampling process (which is a stochastic operation) into a linear combination of deterministic and stochastic elements. An example of this reparameterization trick can be seen on Image 14, where samples z from a normal distribution are rewritten into a linear combination.

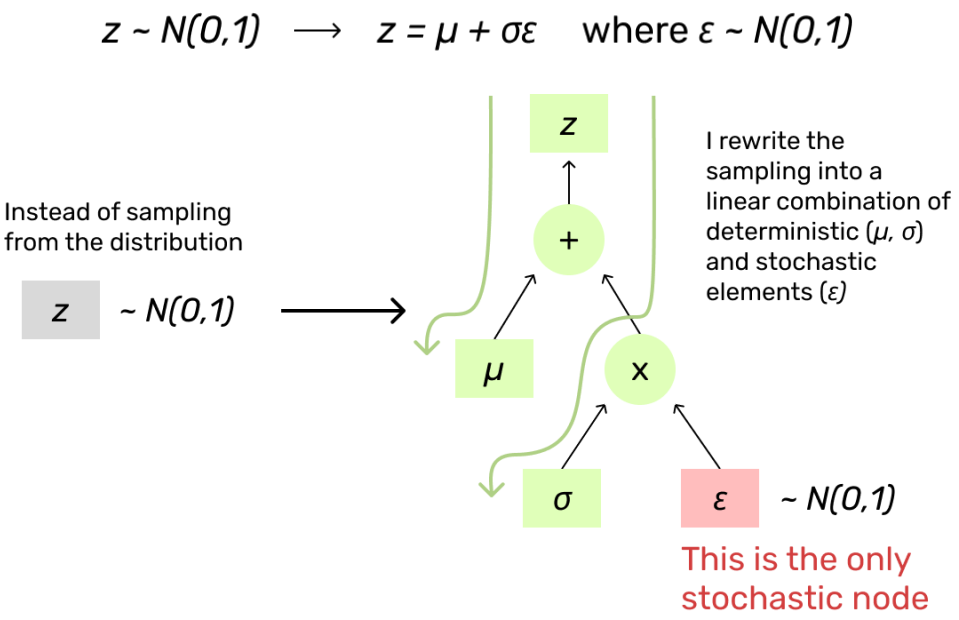


Image 14. The gradient can be propagated along the mean and variance of the normal distribution that yielded sample z

The Normal distribution is continuous, but our distribution of events is categorical, so we then use the **Gumbel-Max trick**. It applies the reparameterization trick (rewriting the categorical sampling process into a

linear combination) by computing the log probabilities of all the classes in the distribution and then adding some noise from the Gumbel Distribution to them.

Then, the argmax function is used to find the class with the maximum value for each sample. The class is then encoded as a one-hot vector to be used in the rest of the neural network (Image 15).

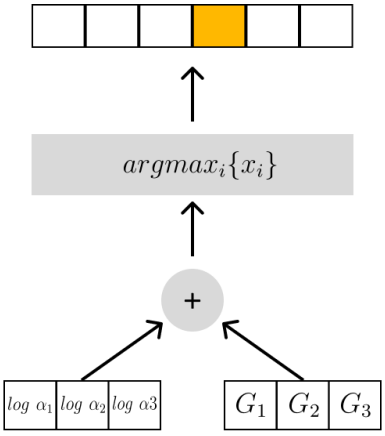


Image 15. Simple diagram of the Gumbel-Max Trick

However, using the argmax operation is still not differentiable, so instead we apply the softmax over the samples and add a λ parameter that determines how close we want our sampling to be with respect to a one-hot-encoding (Image 16). It can be proven that samples from the Gumbel- Softmax distributions are identical to samples from a categorical distribution when λ tends to 0 [23, 24].

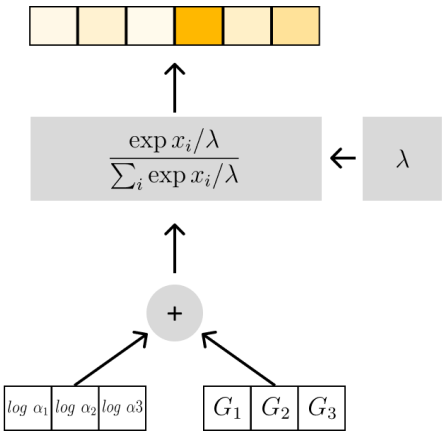


Image 16. Modifications applied to the Gumble-Max Trick to avoid using argmax.

Usage

We are using the Gumbel-Softmax distribution to approximate the sampling process of the discrete MIDI events. We obtain a vector of size 388 for each new event which is very close to a one-hot encoding but has been obtained by differentiable calculations instead of by sampling.

The forward/generative pass is simple, we sample back into a categorical distribution by applying an argmax operation to the "Gumbelized" distribution and we obtain a normal event sequence.

However, in the backward pass, we no longer use the sequence obtained with the argmax but instead we still use the Gumbel-Softmax sample to approximate the gradients, so that backpropagation would still work. This trick is known as "Straight Through the Gumbel-Softmax" (Image 17).

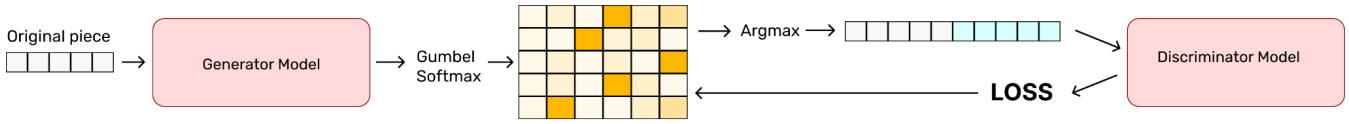


Image 17. Straight Through Gumbel-Softmax

4.6. Training

Our Transformer-based GAN demo has been trained during 24h in the UPC Calcula servers. It required 20Gb of GPU memory + 30Gb of physical memory. In the following section we will discuss the results.

As we were expecting, we observed how the training steps were slow, mainly because in each one of them, the model generates an artificial sequence from a random seed, which is costly in terms of time. During the training, 60 checkpoints of the generator model were stored for future analysis, one for each epoch. However, as a starting evaluation point, we have been testing with the last of these checkpoints, and we discuss these experiments in the following section.

5. Experiments and Results

In this section we will be explaining all the experiments realized with the generator, the discriminator and the closed GAN. We will be reviewing the results and discussing them. All the generated outcomes can be listened to in our Google Drive repository [26].

5.1. Generator experiments

We performed two rounds of experiments for the generator, one with the pre-trained model and another with the one fine-tuned by the GAN. We have generated a small set of artificial pieces using as seeds compositions from different authors and from different historic moments. We have compared the results from both models. These artificial songs can be listened to at [26].

Pre-Trained generator results

The pre-trained generator (mentioned in 4.2.) has achieved a good level of musical abstraction on its creations.

If we focus only on the generation and for a moment forget about the seed, in overall, the pre-trained model achieves great results with coherent pieces, maintaining in good shape the tempo, clef, scale and usually even the rhythm. As the generated sequence grows, there exists an expected performance drop that usually causes the song to slightly lose its original tempo and scale. Nevertheless, the Transformer architecture prevents this drop from being excessive.

Although the composition quality largely varies for each generation, the melodies are usually plain, simple and often repetitive, becoming the weakest point of the model (an example of this at [29]). On the other hand, the model excels at building chords and combining many notes at the same time (an example of this at [30]).

When we listen to the full piece including the original seed, one can usually notice the exact moment when the starting sample ends and the generator continues the sequence. It usually is perceived as an abrupt change, often in the melody (an example of this at [31], it usually happens in all the pre-trained model generations). However, the generator is very

reliable at preserving other characteristics from the seed piece, such as the tempo, the clef and the rhythm (same example [31] or [30]).

Overall, we consider that the pretrained model is a solid starting point to fine-tune the model by means of the GAN.

Fine-Tuned generator results

The differences in the results after fine-tuning the model are small but worth mentioning.

If we look solely at the artificially generated section of the pieces, the model still makes a good job at producing coherent and solid compositions. Although being a small difference, the most noticeable improvement we have found is on the melody. As we mentioned in the previous section, the melody from the pre-trained model lacked complexity and usually was repetitive. We have observed a slight step forward in the fine-tuned model as now it seems to take more risks and propose bigger variations in the melodic facet.
While the songs are not necessarily “better” from our subjective point of view, they look more organic and perhaps more “humane” than the ones from the originally pre-trained model. Some examples of this can be seen comparing generated pieces from the same seed before and after fine-tuning, like [29] with [32].

When taking into consideration the full composition including the seed piece, we find no differences between the pre-trained and fine-tuned model. It is still noticeable when the seed melody ends and the generated one starts. In some cases, taking more risks also makes failures more noticeable, like in [35], where the seed melody is completely lost.

Discussion of the results:

While we feel that some aspects of the fine-tuned model are performing better than the pre-train model, it is true that the improvements are not remarkable. The model seems to generate more organic compositions, which is part of what we were trying to accomplish, but these are not necessarily “better” in terms of musical enjoyment.

Overall, the model seems to assume more risks and that can turn out to become beneficial or to make failure more visible depending on the case.

To find out more about the reasons behind these marginal improvements, we have performed a few sets of experiments to the discriminator model.

5.2. Discriminator experiments

The first experiments performed during the project were on the discriminator. First, we wanted to get an idea about the power of the model to properly discriminate between artificial and original musical pieces. To do so, we used the pre-trained generator (mentioned at the end of 4.2) to provide rich compositions and started training the discriminator using the generations by the model and real data from the MIDI dataset. We expected the discriminator to classify artificial pieces as “artificial” and original pieces as “original”.

The results were poorer than we expected. The discriminator only reached an f1-score of around 0.55 after an entire day of training, meaning that it only performed slightly better than a random classificator. We tried to get better results by modifying the learning rate and other hyperparameters, even by adding more transformer layers of attention heads, but even that didn’t make the model have a significantly better performance (Image 18).

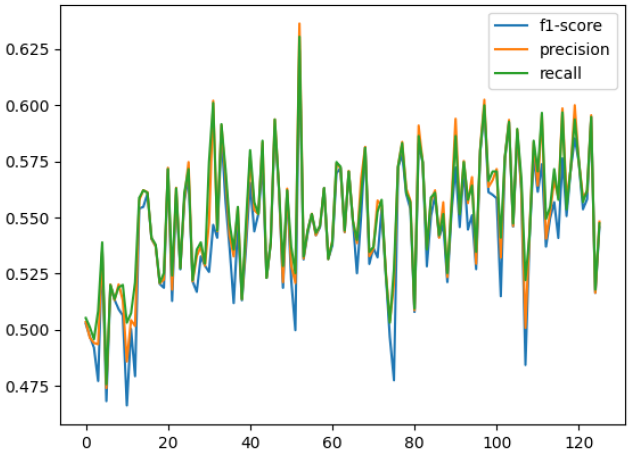


Image 18. F1-score, precision and recall of the discriminator after 24h of training.

To test the capabilities of the architecture and make sure we were not having an implementation bug or an architecture defect, we opted to **add uniform noise into the artificial pieces** in order to increase the apparent differences with respect to the original ones.

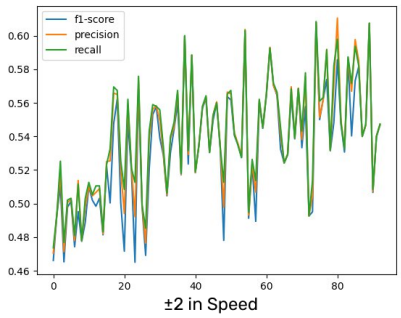
In particular, we opted to slightly randomize the values of the *speed* or the *temporal-shift* events. We did not modify the events of *note-on/note-off*

as these complement each other and not synchronizing them can cause the MIDI file to be corrupted.
We took the predicted speed or time-shift events and added uniform noise within a small range. E.g. If the Time-Shift(50ms) event was found in the generation, we randomly modified its value within a range of ± 20ms.

The goal of this experiment was to make sure that the discriminator architecture actually worked and was indicated for sequence classification. These noisy generations can be listened to at [26].

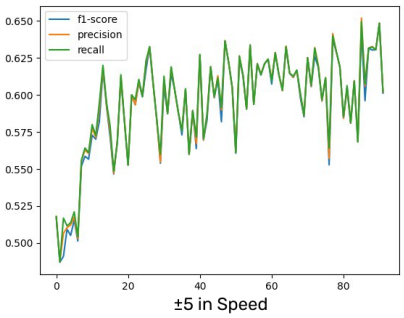
As we added noise to the sequences we rapidly observed how the discriminator was now increasingly capable of discerning between the original (not noisy) and the artificial (noisy) sequences.
We performed two experiment sets, in the first one modifying the Speed and in the the last one modifying the Time-Shift:

Image 19.1



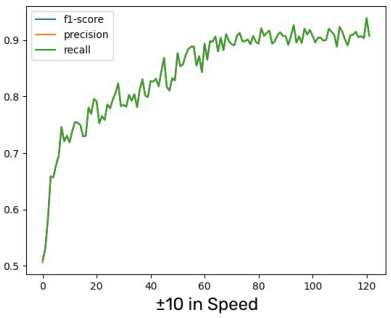
±2 in Speed

Image 19.2



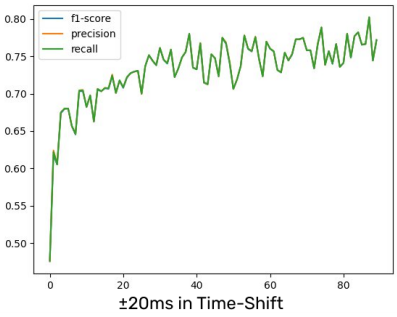
±5 in Speed

Image 19.3



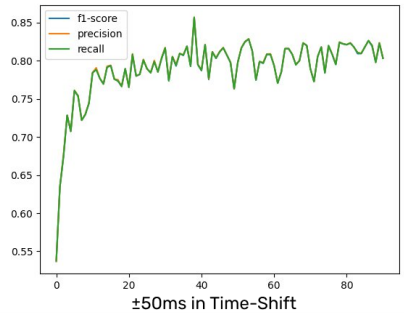
±10 in Speed

Image 19.4



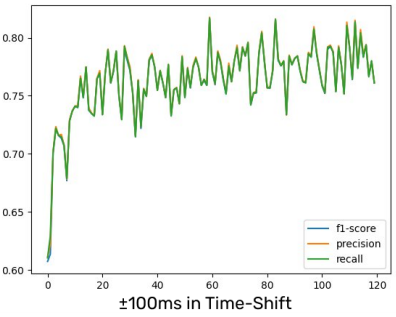
±20ms in Time-Shift

Image 19.5



±50ms in Time-Shift

Image 19.6



±100ms in Time-Shift

Focusing on the sequences where the noise has been applied to the speed events (Images 19.1, 19.2, 19.3]), it seems like the model still has a hard time classifying if the amount of noise is small (Images 19.1, 19.2). However, as the randomness value increases, it gets an outstanding classification score (Image 19.3).

On the other hand, adding noise to the Time-Shifts (Images 19.4, 19.5, 19.6) seems to have a greater impact much rapidly. Even for small amounts of noise around ±20 ms (Image 19.4), the model quickly gets good scores in

the classification. It is interesting to see that adding bigger amounts of noise to the Time-Shift events does not make a bigger difference (Images 19.5, 19.6).

Discussion of the results:

It is interesting to talk about the differences between adding noise to the Time-Shifts and adding it to the Velocity events. It seems reasonable that modifying the temporal dimension, even slightly, can cause the piece to immediately start sounding weird, which can be easily identified by the discriminator. On the other hand, modifying the speed (volume) seems to be less noticeable unless the changes are big.

After reviewing the three experiments, it seems clear that the architecture of the discriminator model is capable of classifying sequences if these are different enough, however, it does not do a good job when trying to differentiate between artificial and original pieces. This can be interpreted in two ways:

1. That the generated pieces are extremely similar to human creations.
2. Or that the discriminator capabilities are quite limited, and it only gets the job done if we help it by adding noise to the sequences.

After performing the experiments on the generator at 5.1, it does not seem likely that we are facing the first case, given that the generations (without being necessarily bad) have significant differences with respect to human compositions. We then assume that we are in the second case, meaning that whether the discriminator architecture or its training method are not powerful enough for the task.

Linking to the generator experiments, it is now clear why the changes between the pre-trained and fine-tuned model are just marginal: because the poor performance of the discriminator is limiting the fine-tuning and improvements of the generator.

6. Conclusions

Three main goals were proposed for this project.

1. Defining a Generative Adversarial Network (GAN) to discern between artificial and human music compositions.

2. To use this GAN to fine-tune the generative model.

3. To empirically assess if the fine-tuned generative model outperforms the original one.

We confidently can say that we have accomplished the first of these goals. We are proposing a model architecture that makes possible fine-tuning a Transformer-based generative model by updating its nodes via backpropagating the loss obtained from a discriminator model. We have also shown how this can be done by means of the Gumbel-Softmax.

The accomplishment of the second objective is blurrier. While it is completely undeniable that we have fine-tuned the generator model, we have not achieved the discriminator to be a proper adversary. Even if the discriminator is doing better than a random classifier, we can only affirm that our fine-tuned generator assumes more risk and often generates more organic compositions.

Finally, as the second objective was directly related with the last one, we can not say that we have succeeded in assessing that the fine-tuned generator is better than the original pre-trained one. Nevertheless, we invite the readers to listen to the artificial generations [26] and compare by themselves.

As the way it looks, the discriminator model has created a bottleneck in our GAN, preventing the generator from being properly fine-tuned. We suggest that finding a more suitable use of the discriminator or a better architecture could have a great impact and make more visible the real potential of fine-tuning the generator by means of a GAN.

6.1. Future research

Even if the objectives of the project have only been partially fulfilled, we are convinced of the approach and highly expect to keep working on it for some time.

We propose two new paths of research based on this project:

- Finding a more suitable discriminator usage and re-train the GAN to see the true potential of the approach.

- Analyzing the current model's compositions, including human evaluation, to identify what are the most visible changes of the generated sequences before and after the fine-tuning.

6.2. Author notes and acknowledgements

Working on this project has been an exciting experience and I expect to continue with this area of research in the following years. While the results have not been as impressive as I would have liked, I feel optimistic about the viability of my GAN approach. I feel confident that there is a better architecture for my discriminator and that implementing it will lead to better results in the next project.

I would like to express my sincerest gratitude to Philippe Salembier Clairon from the Image Processing Department at the UPC for supervising my project with such professionalism yet being so approachable and kind to me.

7. References

[1] Briot, J.-P., Hadjeres, G. and Pachet, F.-D. (2019) *Deep learning techniques for Music Generation -- A survey*, *arXiv.org*. Available at: https://arxiv.org/abs/1709.01620 (Accessed: October 3, 2022).

[2] Hao-Wen Dong, Wen-Yi Hsiao, and Yi-Hsuan Yang. Pyp- ianoroll: Open source python package for handling multi-track piano roll. Proc. ISMIR. Late-breaking paper;[Online] https://github. com/salu133445/pypianoroll, 2018.

[3] Oore, S. *et al.* (2018) *This time with feeling: Learning expressive musical performance*, *arXiv.org*. Available at: https://arxiv.org/abs/1808.03715 (Accessed: October 3, 2022).

[4] OpenAI (2021) *Jukebox*, *OpenAI*. OpenAI. Available at: https://openai.com/blog/jukebox/ (Accessed: October 3, 2022).

[5] Rumelhart, David E; Hinton, Geoffrey E, and Williams, Ronald J (Sept. 1985). Learning internal representations by error propagation. Tech. rep. ICS 8504. San Diego, California: Institute for Cognitive Science, University of California.

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

[7] Vaswani, A. *et al.* (2017) *Attention is all you need*, *arXiv.org*. Available at: https://arxiv.org/abs/1706.03762 (Accessed: October 3, 2022).

[8] Brown, T.B. *et al.* (2020) *Language models are few-shot learners*, *arXiv.org*. Available at: https://arxiv.org/abs/2005.14165 (Accessed: October 3, 2022).

[9] Huang, C.-Z.A. *et al.* (2018) *Music transformer*, *arXiv.org*. Available at: https://arxiv.org/abs/1809.04281 (Accessed: October 3, 2022).

[10] Payne, C.M.L. (2021) *Musenet*, *OpenAI*. OpenAI. Available at: https://openai.com/blog/musenet/ (Accessed: October 3, 2022).

[11] Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D. & Sutskever, I. (2018), 'Language Models are Unsupervised Multitask Learners', .

[12] Goodfellow, I.J. *et al.* (2014) *Generative Adversarial Networks*, *arXiv.org*. Available at: https://arxiv.org/abs/1406.2661 (Accessed: October 3, 2022).

[13] Ho, J., Jain, A. and Abbeel, P. (2020) *Denoising Diffusion Probabilistic models*, *arXiv.org*. Available at: https://arxiv.org/abs/2006.11239 (Accessed: October 3, 2022).

[14] Liu, M.Y. and Tuzel, O., 2016. Coupled generative adversarial networks. Advances in neural information processing systems, 29.

[15] Dong, H.W., Hsiao, W.Y., Yang, L.C. and Yang, Y.H., 2018, April. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 32, No. 1).

[16] Papineni, K., Roukos, S., Ward, T. and Zhu, W.J., 2002, July. Bleu: a method for automatic evaluation of machine translation. In Proceedings of the 40th annual meeting of the Association for Computational Linguistics (pp. 311-318).

[17] Lin, C.Y., 2004, July. Rouge: A package for automatic evaluation of summaries. In Text summarization branches out (pp. 74-81).

[18] Kaelbling, L.P., Littman, M.L. and Moore, A.W., 1996. Reinforcement learning: A survey. Journal of artificial intelligence research, 4, pp.237-285.

[19] Afsar, M.M., Crump, T. and Far, B., 2021. Reinforcement learning based recommender systems: A survey. ACM Computing Surveys (CSUR).

[20] Hawthorne, C. *et al.* (2019) *Enabling factorized piano music modeling and generation with the maestro dataset*, *arXiv.org*. Available at: https://arxiv.org/abs/1810.12247 (Accessed: October 4, 2022).

[21] Ian Simon and Sageev Oore. "Performance RNN: Generating Music with Expressive Timing and Dynamics." Magenta Blog, 2017. https://magenta.tensorflow.org/performance-rnn

[22] Benjaminson, E. (no date) The Gumbel-Softmax Distribution, The Gumbel-Softmax Distribution – Emma Benjaminson – Mechanical Engineering Graduate Student. Available at:

https://sassafras13.github.io/GumbelSoftmax/ (Accessed: October 6, 2022).

[23] Jang, E., Gu, S. and Poole, B. (2017) *Categorical reparameterization with Gumbel-Softmax*, *arXiv.org*. Available at: https://arxiv.org/abs/1611.01144 (Accessed: October 6, 2022).

[24] Jang, E. "Tutorial: Categorical Variational Autoencoders using Gumbel-Softmax." 8 Nov 2016. https://blog.evjang.com/2016/11/tutorial-categorical-variational.html (Accessed: October 6, 2022).

[25] C. J. Maddison, A. Mnih, and Y. W. Teh, "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables," 5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc., Nov. 2016. ArXiv ID: 1611.00712 https://arxiv.org/abs/1611.00712 (Accessed: October 6, 2022).

[26] Pau Lozano, Generated Musical Pieces. https://drive.google.com/drive/folders/1vR0yMYjYJNt-5Nxze27wSMcIPM7loXFT?usp=sharing

[27] Chintapalli, K. (2019) *Generative adversarial networks for text generation - part 3: Non-RL methods*, *Medium*. Becoming Human: Artificial Intelligence Magazine. Available at: https://becominghuman.ai/generative-adversarial-networks-for-text-generation-part-3-non-rl-methods-70d1be02350b (Accessed: October 8, 2022).

[28] Kusner, M.J. and Hernández-Lobato, J.M. (2016) *Gans for sequences of discrete elements with the gumbel-softmax distribution*, *arXiv.org*. Available at: https://arxiv.org/abs/1611.04051 (Accessed: October 8, 2022).

Generated pieces:

[29] PRE-TRAINED, Bach, Toccata in E Minor, BWV 914
https://drive.google.com/drive/u/3/folders/1SYycjaaoTZIoUWs7cgG2y_5PPbl6Quxk , the artificial generation starts at second **45**.

[30] PRE-TRAINED, Schubert Piano Sonata No18 in G Major, D.894
https://drive.google.com/drive/u/3/folders/1cbOz7d0qEkf5gynZo7iACZgog204ZXbZ , the artificial generation starts at second **31**.

[31] PRE-TRAINED, Chopin, Ballade No. 4 in F Minor, Op. 52
https://drive.google.com/drive/u/3/folders/1N5Ebi5m69Yx2SvI3gqgrezRr0sTRC4tJ , the artificial generation starts at second **21**.

[32] FINE-TUNED, Bach, Toccata in E Minor, BWV 914
https://drive.google.com/drive/u/3/folders/1SYycjaaoTZIoUWs7cgG2y_5PPbl6Quxk , the artificial generation starts at second **45**.

[33] FINE-TUNED, Schubert Piano Sonata No18 in G Major, D.894
https://drive.google.com/drive/u/3/folders/1cbOz7d0qEkf5gynZo7iACZgog204ZXbZ , the artificial generation starts at second **31**.

[34] FINE-TUNED, Chopin, Ballade No. 4 in F Minor, Op. 52
https://drive.google.com/drive/u/3/folders/1N5Ebi5m69Yx2SvI3gqgrezRr0sTRC4tJ , the artificial generation starts at second **21**.

[35] FINE-TUNED, Liszt, Venezia e Napoli, S.162 (Complete)
https://drive.google.com/drive/u/3/folders/14m69Xk62on8_DNNqt53Xka2Lc8ATk7Ih, the artificial generation starts at second **38**.