Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Tuning RNN-based musical composers for specific compositional styles using Deep Q-Learning

Author:
Rodrigo Schönburg Carrillo de Mira

Supervisor:
Björn Schuller,
Eduardo Coutinho

Submitted in partial fulfillment of the requirements for the MSc degree in Advanced Computing of Imperial College London

September 2018

Abstract

Music composition is a complex field which is difficult to automate because the computational definition of what is good or aesthetically pleasing is vague and subjective. Many neural network based methods have been applied in the past, but they lack consistency and in most cases their outputs fail to impress.The most common issues include excessive repetition and a lack of style and structure, which immediately label the compositions as artificial. In this project, we build on two successful models created by Magenta - the Melody RNN and the RL Tuner - combining them and extending them to emulate a specific musical genre - *the Galician Xota*. To do this, we design a new rule-set containing rules that the composition should follow in order to adhere to this style. We then implement them using reward functions, which are used to train the Deep Q Network that will be used to generate the pieces. After extensive experimentation, we achieve a successful implementation of our rule-set and outline a solid research methodology for future researchers looking to use this architecture. Finally, we propose some promising future work motivated by two proofs of concept.

Acknowledgments

I would like to thank Eduardo Coutinho and Björn Schuller for their supervision as well as Georgios Rizos for his help with technical and conceptual issues during my research. I would also like to thank Emilia Parada-Cabaleiro for her help in choosing and gathering an adequate dataset for the development of the project. Additionally, I would like to thank my friends and family for their continued support without which this work would not have been possible. As a final note, I would like to show my appreciation for David Tuckey's help and encouragement during the past 12 months.

Contents

Chapter 1

Introduction

Since the birth of Artificial Intelligence (AI), there have been many attempts to automate creativity in various areas. However, the most elusive field has always been music, due to its deep connection with human emotions and our limited understanding of how it can affect us so profoundly with a simple sequence of notes. Algorithmic approaches for this problem have been proposed for various decades, but they cannot truly be considered intelligent since they operate on a list of commands made by human beings, instead of learning on their own. With the advent of Deep Learning and its exponential increase in popularity during the twenty-first century, a promising method has been found for composing music: recurrent neural networks. Specifically, Long Short-Term Memory (LSTM) networks [1] have been widely used due to their superior ability to model long term dependencies accurately. Recently, research laboratories such as Google Magenta have managed to build around this type of network to create more complex architectures. These attempt to perfect the usage of RNNs while pairing them with other techniques such as Reinforcement Learning (RL) [2]. On the other hand, some research groups have focused on using Generative Adversarial Networks (GANs) to exploit LSTMs [3], while others have even attempted to compose music without using any type of RNN [4].

In any case, the task of generating music with no human interaction is inherently difficult. Generating images using neural networks, for instance, is generally not an easy task, but when the results are not ideal it is easy to describe why: the image is too blurry, this object should not be in the picture, among others. This is due to the transparent nature of images as something that we can grasp and describe easily. Music, on the other hand, is a much more cryptic medium. While we find it very easy to know whether or not we like a certain song, it is very difficult to describe why and even more challenging to actually theorize a song that would match our taste (which is typically a very narrow slice from the set of possible compositions). This makes it very difficult for the neural network to create pleasing results, given that its margin for error is so minute, and leads to an arduous tuning procedure, since it is hard to know how to steer the artificial composer in the correct direction.

Previous works in artificial music composition have struggled due to multiple specific issues. The first of these is overfitting. RNNs tend to overfit if not monitored

correctly and trained with preventive measures such as a high dropout rate. This is especially problematic in music composition due to the limited datasets available. The main issue with RNNs, however, is the inability to live up to the promise of taking into account all previous inputs. Even with the introduction of the LSTM cell and attention mechanisms, there are still issues with numerical instability which prevent it from being able to model all of the previous composition (in the case of music generation) with the current state of the network. In addition, more recent methods such as GANs (Generative Adversarial Networks) and VAEs (Variational Auto Encoders), which have attained massive success in other generative tasks, have failed to impress with regards to composing pleasing music, remaining a step behind RNNs.

This project [1] focuses on emulating a very specific compositional style (*the Galician Xota*) by applying a new dataset and rule-set (set of music theory rewards) to an existing project: the RL Tuner [2]. This model focuses on Reinforcement Learning, attempting to use a trained RNN and a set of music theory rules to provide rewards in a Q Learning architecture, with the intent of tuning the compositions of traditional LSTMs. We also attempt to combine combine the RL Tuner with the Attention RNN [6], another promising project that instead focuses on imbuing traditional LSTM's with attention mechanisms, which have gained tremendous popularity in other areas regarding machine learning. These projects both include experiments which boast very impressive outputs. In the end, the research procedure leads to experimentation with the new rule-set in order to transfer the style of this genre to RNNs which were trained with a general dataset, so as to show the power of the RL Tuner. Our contributions are as follows:

Major contributions:

- Developing a new rule-set (set of rewards) for the Galician Xota genre and integrating it in the RL Tuner code.
- Developing a new version of the Basic RNN which adds attention mechanisms while keeping the same encoding and modifying the RL Tuner in order to accept the checkpoints trained with this new model.
- Experimenting with a broad set of RNN configurations and reward modes in order to provide a more complete set of detailed experimental training procedures which can aid further research with the RL Tuner.
- Describing a set of empirically justified findings which dictate how The RL Tuner can be used and extended while explaining its limitations and how they can be compensated.
- Implementing simplified demonstrations of two extensions that can be made to this project - Inverse Reinforcement Learning and emotional rule-sets - in order to motivate future research with this model.

[1] The code and results for all of the experiments in this project, as well as the full dataset are available on https://github.com/miraodasilva/RL_RNN_MusicComposer and will be referenced throughout this report as [5]. **The generated samples can also be found in the repository [5]**

Minor contributions:

- Repairing and extending the original RL Tuner architecture in order to enable a more complete research process.
- Creating an open-source Guitar Pro 5 to MIDI converter which can be used in other projects with MIDI based datasets.
- Implementing an open-source data augmenter for MIDI datasets based on previous research [7].

This report will feature six remaining chapters (excluding the introduction). Chapter 2 will be dedicated to background knowledge, explaining the relevant previous work in artificial music composition with Deep Learning, while also exploring Deep Q Learning and basic music theory. Chapter 3 will focus on the Melody RNN (An RNN Model published by Magenta) and its three variants: the Basic RNN, the Lookback RNN and the Attention RNN. We will then explain how we used attention mechanisms and dealt with overfitting during the training procedure, as well as some further details regarding specific parameters. Chapter 4 will focus on the main object of our research, the RL Tuner, and how it was configured. Specifically, we mention the original as well as the new rule-sets, mentioning each of the rules that are contained in them and the reasons for their design. In chapter 6, we perform a wide range of experiments and proceed to analyze them by evaluating how the outputs were affected by each rule-set. We focus on one configuration with more detail, while still analyzing every experimental procedure in order to perform a complete study of the RL Tuner. Finally, in chapter 7, we summarize the work done in this project and explicitly denote the findings that can be concluded after the experiments and analysis mentioned in previous chapters. We finish by suggesting future developments for this project and proceed to prove their viability by performing two proofs of concept.

Chapter 2

Background

In this chapter we will explain the technical and domain knowledge which will be necessary to understand this project in its entirety. We begin by mentioning the origin of neural networks, specifically RNNs and LSTMs. Then we move on to speak about previous attempts to apply Deep Learning to music composition, discussing their achievements and limitations. We move on to speak about GANs, summarizing the four main projects that have used GANs for music generation. We mention some other relevant projects in this area, and conclude by defining the Markov Decision Process for this project and explaining the algorithm behind Deep Q Learning and how it will be applied. Finally, we define some fundamental music theory concepts which will be essential to understand the original work presented later.

2.1 Overview of recent work in Deep Learning, Reinforcement Learning and artificial music composition

2.1.1 Recurrent Neural Networks

The first and most evident approach for artificial music composition has been through the use of recurrent neural networks (RNNs). This type of network played a substantial role in the Deep Learning revolution by providing a way to model complex sequences with various dependencies. The building block for this kind of architecture was, of course, the research performed by Hopkins [8] during the 1980's, which introduced neural networks to the world of computing.

In any case, the first network that can be accurately compared to modern RNNs was brought to life by Elman, simplifying the previous architecture by Jordan [9]. Its structure was based on a traditional feedforward network, with the addition of a recurrent connection between the current hidden layer and the previous hidden layer (considering the sequence as a series of time-steps). This weighted edge provides a much needed context which is passed between time-steps and allows for the modeling of dependencies through time. In simpler terms, this makes it so that the output

at a certain point in the sequence is dependent not only on the input at that time but also on the previous inputs. Elman demonstrated that this network could indeed be trained to learn long-term dependencies relatively well. The training was done using backpropagation, which begins by providing an input and the expected output (according to data) and propagating the input forward with the current weighted edges. Then, the observed output is compared to the expected output using a loss function which produces an error. This error is propagated backwards (hence the name of this method), performing gradient descent on the loss with respect to the weights of the edges. In this case, the algorithm used is specifically backpropagation through time, which expands on the previous concept by propagating the error through time.

As the popularity of Recurrent Neural Networks grew, the extent of their limitations became increasingly apparent. Although these kind of architectures were rapidly applied to sequential tasks such as Natural Language Processing and even music generation [10], it was clear that none of the current variants could accurately maintain a state through time which could consistently model long term dependencies between sections of the sequence. The necessity for an improvement in this regard led to the creation of the Long Short Term Memory Cell [1]. The general idea behind LSTM Cells is that they allow for the propagation of the state of the network by not directly subjecting it to an activation function at every time-step. Instead, the state flows and is altered by the input through three gates: a forget gate, an input gate and an output gate (1, 2 and 3 in figure 2.1). This design means that the Cell itself can model the varying importance of the previous inputs through an ever-changing state which can be tuned with much more detail. In this way, it was possible to structure Recurrent Neural Networks around these new cells for better performance in sequential tasks.

With this substantial leap in performance for RNNs came, of course, better performance for artificial music composers. This started with the first attempt to use LSTM Networks for music composition by Eck [12]. As there were no previous attempts at the subject, this study set a lot of the standards for AI in music generation. The experiments are performed using LSTM's optimized with a cross-entropy function. The data is composed of one-hot vectors representing each time-step where 1 means that the note is being played and 0 means it is not. Chords are allowed to contain a range of 12 notes and melodies have a range of 13 notes. This is a simple yet reliable data representation which is not very distant from the one used in state of the art projects currently. It is worth noting that chords are represented through their notes and not symbolically, i.e., a C major is not represented as a 1 in a one-hot vector, but as C=1 E=1 G=1. This is a bold choice since it expects the network to learn how to form chords instead of selecting one of them from a predefined list. The conversion between standard annotated formats such as MIDI and this one hot vector is not clear in this paper, which could imply that the one-hot vectors were made manually.

In this study, the authors carried out two experiments. The first aims to simply learn chords, while the second aims to learn chords and melodies. The training

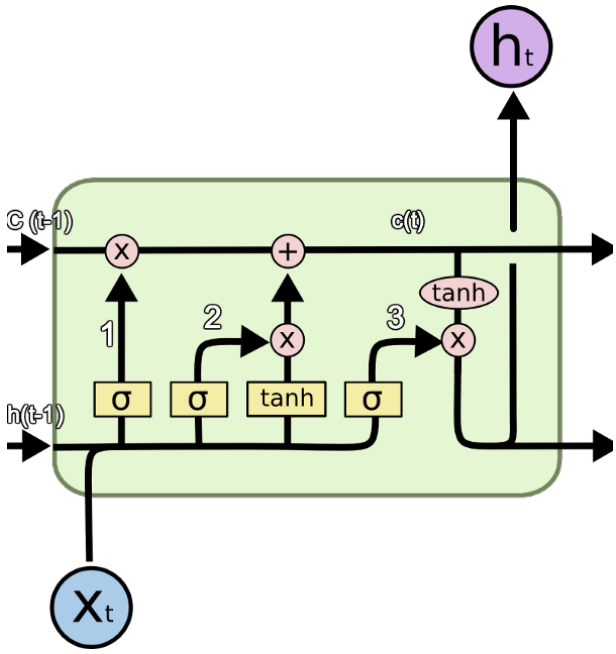


Figure 2.1: Schematic representation of the LSTM Cell, where c(t) represents the state or memory of the cell, h(t) is the output, σ and $tanh$ represent activation layers, x_t is the input of the cell and h_t is the hidden output. [11]

data is composed of chord/melody examples in the blues style formulated by hand with random variants. This clearly shows that this research is attempting to recreate musical sequences with slight noise and not actually attempting to simulate creativity, which constitutes a very important distinction. The first experiment works well, with the full chord sequence (which is present in all of the data) being learned with relative ease with a single layer LSTM (one input layer, one hidden layer and one output layer). The second experiment learns the chord sequence first and, once it is learned, composes music freely. Again, the data is quite restrictive but the melodies generated model the training pieces quite well. In the end, this paper sets realistic goals and achieves them with impressive consistence, marking a big step for the area of music composition using AI. In the following years, LSTM's slowly took over in areas requiring the modeling of complex sequential tasks such as Speech Recognition or even Protein Homology Detection. The following paragraphs shall focus on the most interesting recent projects in artificial music composition using LSTM networks.

Google Magenta is a lab tasked with recreating art using AI, and many advances in this area have been attributed to its members. This is mainly due to the fact that they have created a solid open-source framework for artificial music composition, but their experimental models for this purpose are what make them a recurring name in the research world. Of these models, there are two which seem to be particularly interesting for the future of AI. The first of these is the Reinforcement Learning Tuner [2]. This project aims to build upon the traditional LSTM architecture for music composition mentioned previously with some previously determined human knowledge about music theory. Specifically, the model uses (Double) Q-Learning (among

other RL variants such as Psi-Learning or G-Learning) to explore a state space in which the actions are the new notes, the states are the previous notes and the rewards are determined by a mixture of the probability of the new note according to the (pre-trained) LSTM network and its adherence to general music theory rules , in this case directly taken from Gauldin [13] and implemented manually. This leads to substantial improvements on traditional RNN outputs, with pieces sounding much more reasonable and consistent. This statement gains a somewhat objective quality since the outputs generated by this architecture were much preferred to the pre-RL generations, according to a study performed on Mechanical Turks.

The second model requires some background knowledge. As stated earlier, LSTM's were meant to improve on the traditional RNN's ineptitude for modeling long term dependencies by using a new type of cells which allowed for the state to pass more smoothly and to be easily fine tuned between time-steps. However, the fact of the matter is that due to the nature of backpropagation, it is common to come across issues such as exploding or vanishing gradients. In short terms, this issue occurs when the gradient of the activation function is too large or too small and it is multiplied repeatedly in order to find the gradient to perform gradient descent on the weights of the network. This means that the gradient descent will either become too steep or too slow, which will harm performance either way and cause the state to not be influenced fairly by all of the previous inputs in the sequence. This issue combined with the fact that the design of the LSTM cell is not perfect (does not guarantee a state which is truly representative of the previous input) once again kickstarted a search for more accurate ways of capturing long term dependencies. This search once again led to an iconic paper, which introduced attention mechanisms.

The paper by Bahdanau [14] was elaborated in the context of machine translation (translating from one human language to another such as English to French) using neural networks, in this case an encoder-decoder architecture. The idea was to use a context vector for each time-step which was influenced by every other time-step. Specifically, this vector was a sum of annotations, one for each time-step, with trainable weights associated with them. These annotations are the concatenation of a forward hidden state and a backward hidden state calculated by a bi-directional LSTM network, and are meant to represent the state of the network at that time-step. This allows for the network to "pay attention" to any of the previous states of the network in order to generate the output. This is useful in translation, for instance when we want to translate the word "the" to French, it is useful to check the next word for its gender, to know if we should output "le" or "la". Some researchers found this idea so impressive that they followed up on it, claiming it would replace RNNs altogether [15].

This concept was adopted by Google Magenta to produce the Attention RNN [6]. In this case there is no encoder/decoder, the LSTM network works in the same way but it uses an attention mechanism to take into account previous inputs in the form of a context vector. This is a somewhat simplified version of the original concept, but it

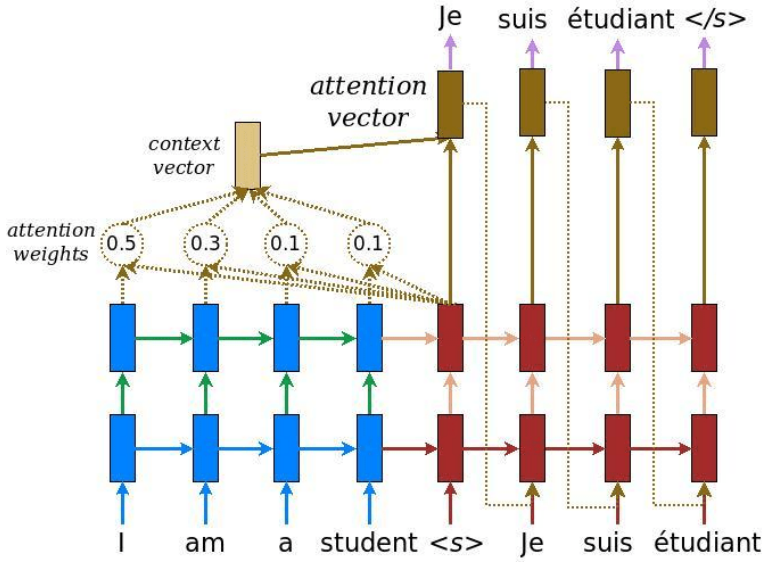


Figure 2.2: Scheme explaining the use of attention mechanisms for the original encoder-decoder architecture used for neural machine translation. [16]

works to great effect, producing melodies that seem to have more structure, following up on the priming melody as a sort of motif. It is worth noting that this project also produced another model called Lookback RNN, which is a more brute-force way of taking into account previous elements of the composition, and (arguably) produces less interesting outputs.

Apart from Google, some very interesting projects focusing on LSTM's for music have appeared very recently. The first of these [17] is inspired by a Youtube video titled "Songs From Pi" and introduces some interesting contributions. This project aims to compose pop songs hierarchically by composing a melody, then chords conditioned on the melody and then drums, also conditioned on the melody. This makes for four layers, two of which are dedicated to the key being pressed and the duration of the press during the melody. The network used for the melody generator is a two-layer Lookback LSTM (as introduced by Google magenta [6]) which is also conditioned on a scale randomly sampled from 4 common pop scales and a melody profile. The melody profile is perhaps the most exciting aspect of this paper, but it is not given much attention by the authors. It attempts to model the general flow of the piece by classifying every two bars of it as one of ten clusters via k-means. Therefore, we get a sequence of one-hot vectors which can portray the ups and downs of the song. *This is an extremely valuable idea since it is attempting to imbue music with a specific structure, which is a quality most artificial composers are severely lacking.* The architecture is trained on over 100 hours of pop music and the outputs generated are generally pleasing, while not exactly revolutionary.

Another recent paper which presents interesting concepts in this area is Jambot [9], which composes polyphonic melodies accompanied by chord progressions. This architecture uses a chord LSTM to compose a chord progression, which then conditions

the melody LSTM to compose polyphonic music freely. Apart from the polyphonic aspect, this is not novel content. The dataset used is a heavily modified version of the widely used Lakh repository [18]. The pieces that are kept are the ones adhering to major/minor scales, and all of these are shifted to the root note of C. This makes for a more consistent dataset but conditions the generation to compose in C major/ C minor, which may be seen as slightly oppressive. The chords are extracted by analyzing the three most frequent notes in each bar and assuming that to be the chord applied. This is also questionable since it excludes non-triad chords and constitutes a substantial approximation. The interesting aspect is that these chords are represented as word embeddings in Natural Language Processing, which means that only the most frequent triads are given an ID and can be visualized in relation to each other. This leads to the most exciting conclusion of this paper which is that the network forms the proper triads and it is able to extract the circle of fifths, which can be visualized through the "chord embeddings". This shows the power of artificial composers and proves that, in this field, ambitious goals can lead to remarkable findings.

The third paper that is worth noting is DeepBach [19]. This project is substantially more focused than the previous ones, since it attempts to emulate a very specific style of songwriting: Bach chorales. These consist of three voices that accompany a main melody, which were written by hand by Bach (and other musicians) to create layered harmonies in musical pieces. One of the main contributions presented here is the novel data representation, which is more complex than the traditional Piano-roll or even the Magenta encoding. Firstly, it divides the melody into four voices, which is evidently necessary. It also introduces a boolean which is set to 1 if there is a fermata symbol over the current note (indicating that it should be prolonged) and it identifies each time-step with its beat subdivision (an integer between 1 and 4). The architecture used here is quite complex. Two deep recurrent neural networks sum up past and future time-steps, and their outputs are concatenated with the output of a neural network with one hidden layer which takes the current time-step as input. This vector is then fed into a neural network identical to the previous one, which produces our note probabilities for each voice. Gibbs sampling is then applied to generate the voices, choosing a random time-step and a random voice and sampling from the conditional probability distribution $p_i(V_i^t|V_{i,t})$ given by the neural network. This intricate method means that we can re-harmonize any melody with very convincing results, which can be tested by using the simple software developed to make this process easier for musicians.

A note about gradient descent and the Adam optimizer:

Gradient descent [20, 21] is the method with which we optimize the weights of a neural network. Specifically, if we are trying to optimize a function f with respect to its parameter θ, we can perform gradient descent by updating the parameter θ in the following way:

$$\theta_{t+1} = \theta_k - \gamma_k \nabla f(\theta_t) \tag{2.1}$$

where $\gamma_t >= 0$ represents the learning rate (or stepsize), which determines how much the parameter is affected by one step of the descent. To smooth out this descent we can apply a method known as momentum. This involves taking into account previous gradients when updating the current parameters, which is very useful when the gradient is only estimated. The addition to the original formula can be seen below:

$$\theta_{t+1} = \theta_t - \gamma_t \nabla f(\theta_t) + \alpha_t(\theta_t - \theta_{t-1}) \tag{2.2}$$

where:

$$\theta_t - \theta_{t-1} = -\gamma_{t-1} \nabla f(\theta_{t-1}) \tag{2.3}$$

$$\alpha_t \in [0, 1] \tag{2.4}$$

In this new equation, α_t controls the momentum of the descent. There is, however, a third major improvement that can be done over this method. Calculating the gradient of the loss when using Recurrent Neural Networks, for instance, can be computationally expensive. This means that by using the whole dataset for each step in the gradient descent we have to calculate this gradient for every element of the dataset, which can make this process extremely slow. The idea behind stochastic gradient descent is to sample a mini-batch of the dataset (the size of which we can set as a hyperparameter) and using this as an estimate for one gradient descent step instead, in order to make steps which may not be as well guided or steep, but which can be done more quickly. This method, combined with momentum, can lead to great results since the momentum compensates for noisy gradient estimates and the descent can proceed at a faster computational pace.

As statistical methods have developed in the context of Artificial Intelligence, the concept of gradient descent has been perfected and extended to multiple specific optimization methods. The algorithm which we will use in this project and which has been used extensively in Deep Learning is the Adam optimizer [22, 23]. This algorithm encompasses two substantial improvements to the previously mentioned stochastic gradient descent. The first of these is using an adaptive learning rate for each of the parameters (weights of the network). This means that the weights which affect frequent features are given a lower learning rate than weights which affect rarer features. This is beneficial if we are dealing with sparse gradients, which can be the case in Natural Language Processing and, of course, music composition. The second improvement can be described as a more complex momentum mechanism. Specifically, the descent takes into account past gradients (m_t) as well as past gradients squared (v_t), which allows for a more controlled momentum applied to the descent. These values are calculated as such [22, 23]:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.5}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.6}$$

However, these variables are initialized to zero and are therefore biased towards that value. This typically happens when β_1 and β_2 are close to 1, which means that

the updates to these variables at each time-step are rather small, forcing them to stay around their initial value of zero. To counter-act this, we apply the following transformation, which will raise the values of m_t and v_t when β_1 and β_2 are close to 1 [22, 23]:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.7}$$

$$\hat{v}_t = \frac{m_t}{1 - \beta_2^t} \tag{2.8}$$

Finally, these variables are combined in the final update formula for the weights (θ_{t+1}) [22, 23]:

$$\theta_{t+1} = \theta_t - \frac{\gamma}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{2.9}$$

where epsilon represents a very small value to avoid division by zero, which would interrupt the computational process. This algorithm is shown in [22] to be quite effective, hence its generalized use in Deep Learning.

2.1.2 Generative Adversarial Networks

During current times, it is nigh impossible to study generative methods in Deep Learning without coming across Generative Adversarial Networks [24]. This architecture was developed by Ian Goodfellow et al. in 2014 and since then it has become a staple for image generation and has recently been gaining popularity in other areas as well, such as music generation. The general concept that supports this framework is simple: one network generates a realistic output (generator) and the other network attempts to distinguish between this output and real data (discriminator). This makes for a game of cat and mouse which is ruled by a general min-max function [24]:

$$min_G max_D V(D, G) = E_{x \sim p_{data}(x)}[logD(x)] + E_{z \sim p_z(z)}[log(1D(G(z)))] \tag{2.10}$$

Gradient descent is then performed on both networks using this function, meaning that the generator function G must be differentiable, which is an issue for discrete outputs as will be mentioned later. This descent supposedly leads to the global minimum, meaning that the distribution of the generator is equal to the distribution of the data in order to perfectly fool the discriminator. This is often not the case and, although the local minimums can lead to successful generators, this network has severe issues with mode collapsing, which consists of focusing on fooling the discriminator on a limited number of classes of data and then only generating those classes. This is evidently an issue because it does not model the whole distribution and therefore the generator becomes a warped version of what is intended. Regardless, GANs have experienced massive success in image generation because they stray away from the traditional blurry outputs of regular networks and generate concrete images. This happens due to the nature of this network: the generator must focus on specific aspects of the data in order to fool the discriminator, which means it cannot

keep producing blurry, safer outputs.

Regarding the previously mentioned issue about non-differentiable generator functions for discrete outputs, Goodfellow [25] mentions three distinct solutions:

1. REINFORCE algorithm [26].
2. Using the concrete distribution [27] or Gumbel-softmax [28].
3. Training the generate to sample continuous values that can be decoded to discrete ones (e.g., sampling word embeddings directly).

These three approaches are essential to fully understand how to model discrete sequential data using Generative Adversarial Networks.

The first attempt at music composition using GANs lies in SeqGAN [29]. This project focuses on a broad range of sequential applications, having only a small section dedicated to music composition. However, the aspect of it that makes it an interesting and essential piece of research is how it implements the first solution mentioned above: the REINFORCE algorithm. In this paper the generator is an LSTM network and the discriminator is a convolutional neural network, which are fairly reasonable choices to model sequential input. The trick applied here is based on modeling the generator as a stochastic differentiable policy which is rewarded based on the discriminator output. This represents our min-max function and we therefore perform the gradient descent using the policy instead. Evidently, this involves calculating Q values (state-action values), which are computed using Monte Carlo search. The complete algorithm works by initializing the generator and the discriminator using Maximum Likelihood Estimation (MLE) and Cross-entropy respectively, and then performing policy gradient descent. The results are tested using an oracle, which is a representation of the actual data distribution that can then be easily compared to the generator distribution to gauge its optimality. The experiments performed on music generation use the Nottingham dataset (consisting of 1000+ folk music pieces) and manage to outperform traditional MLE. Not many details about this experiment are mentioned since it is clearly not the focus here, but this paper is nevertheless relevant due to its interesting methodology for dealing with discrete outputs in GANs

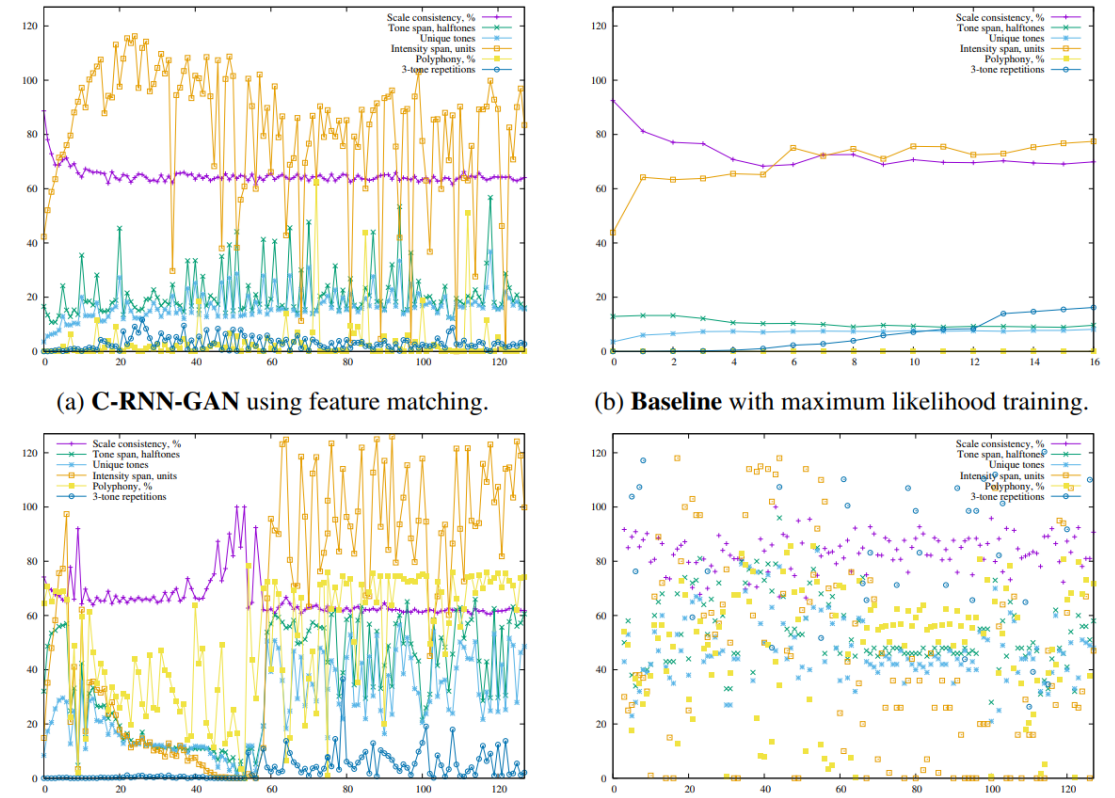
Following this paper came C-RNN-GAN [3], focusing fully on music composition. The architecture is again quite simple: the generator is a Deep LSTM network and the discriminator is a bi-directional Deep LSTM, which is a solid choice since the discriminator is a classifier and can afford to be bi-directional. The data representation used is arguably novel and consists of four identifiers per time-step: tone length, frequency, intensity and time spent since the previous tone. The experiments consist of comparing the outputs to the data and to the baseline, which uses an LSTM with MLE. Backpropagation through time is used, which is expected, but also freezing [30] and feature matching [30]. Freezing consists of stopping updates to the Discriminator (D) when its training loss is less than 70 % of the Generator's (G) training loss. This is a very effective way to keep the training of G and D balanced,

which is crucial for successful results with GANs [25]. Feature matching, on the other hand, means adding regularizers to the min-max equation to ensure that the generated distribution resembles the original distribution, preventing the previously mentioned problem of mode collapsing to a certain extent. One final relevant aspect about the training is that a (very simplified) form of curriculum learning [31] is applied, in the sense that the training begins with short snippets of MIDI and increases their length as the training advances.

The outputs are evaluated using a somewhat arbitrary set of measures (as usual), but one of them - 'Repetitions', measuring the amount of subsequences which are repeated - led to some very revealing results. As we can see in figure 2.3, this metric is very low for all of the generated outputs but it is substantially higher for the real data. *This leads us to conclude something very apparent when listening to samples from various projects in this area: the music lacks a motif which is repeated and modified throughout the song, providing a feeling of consistency and structure.* This is a very important issue, which should be addressed if we expect to produce lengthy outputs which resemble human composed music. In the end, the outputs of this project are comparable to standard LSTMs but reportedly sound more interesting, which might have to do with GANs collapsing to a specific style (as in the blurry images example mentioned earlier).

Perhaps the most groundbreaking piece of work using GANs for music composition is MIDINet [4]. This is due to the fact that it relies fully on convolutional neural networks to generate music pieces, which is largely unprecedented. The general architecture is based upon DCGAN [32], perhaps the most influential architecture for GANs, which is shown to be quite effective at generating images and has some notable features such as vector arithmetic for images. MIDINet uses CNNs for the generator and discriminator, based on this architecture, but also involves a conditioner CNN. It also uses the bar (which is a set of notes, defined in section 2.2) as its compositional unit, whereas traditionally a sixteenth note time-step is used. The generator consists of fully connected layers followed by transposed convolutional layers which "upscale" the one-dimensional input, while the discriminator consists of a few convolutional layers followed by some fully connected layers. The conditioner CNN is essentially the reverse of the generator, downscaling a piano-roll chunk (2D) into a one-dimensional condition which can then be fed into the generator.

In summary, the generator can be conditioned through the one-dimensional it gets from the conditioner CNN and through adding a one-dimensional conditioning vector to its layers [33], which means it can be conditioned on two different inputs (the same is done for the discriminator). This leads to an experiment in which the generator and discriminator are trained with the previous bar as the 2-dimensional condition and the current chord as the one-dimensional condition (a one-hot vector representing the chord symbolically). It is worth mentioning that feature matching is employed (which could be crucial for successful results) and that, unlike RNNs, the generator can compose without a priming melody to get the network started, instead



(a) **C-RNN-GAN** using feature matching.

(b) **Baseline** with maximum likelihood training.

(c) **C-RNN-GAN-3**, using feature matching and three tone outputs per LSTM cell.

(d) The same statistics for a selection of real music in the dataset.

Figure 2.3: Graphs comparing various evaluations measures between artificially generated and human composed musical pieces.[3]

using a randomly sampled one-dimensional vector as an initial input. This process emulates the recurrent aspect of a traditional LSTM with resounding success, leading to results which, according to a user-study, are comparable to the three Melody RNN variants proposed by Google Magenta. This is extremely impressive since this approach represents a completely new direction for the field of music composition using deep learning and already boasts promising results.

Following MIDINet's exciting results came its successor, MuseGAN [18]. This paper builds on the core concept of MIDINet and expands it to compose multi-track polyphonic music with a distinctly modeled structure. Multi-track interdependency is, of course, a very important aspect in music with various instruments playing simultaneously. Here it is represented in three different ways: the jamming model, which has various generators compose melodies which are then judged by their respective discriminators; the composer model with one generator composing all the tracks and then evaluated by one discriminator; and the hybrid model with various generators evaluated by one discriminator. An attempt at producing music with consistent temporal structure is made by conditioning the bar generation on a vector which contains temporal and rythmic information. This results in two temporal models:

generation from scratch, which maps a noise vector to a sequence of temporal context vectors; and track-conditional generation, which generates these same vectors based on a human-composed track, attempting to continue the musical piece. In the end, this leads to a very complex global architecture that deals with vectors modeling intra-track or inter-track information which is dependent or independent of time.

The experiments are performed using the Lakh Dataset [18], which is extended by adapting external data to its format. The music is composed for five instruments: bass, drums, guitar, piano and strings. The outputs are evaluated using some fairly generic metrics such as ratio of empty bars, and also some interesting ones such as tonal distance [34], which describes the harmonicity between two tracks. The results sound quite decent and the experiments made show that batch normalization is essential and that there may be an issue with the way inputs are binarized (which is caused by using the third solution mentioned by Goodfellow [25]). A user study is conducted, indicating that the hybrid and jamming models are the most pleasing. In conclusion, MuseGAN is an extremely ambitious project, which ends up fulfilling its general goals with surprising success, even if the final outputs are very far from what is expected from human composers (whereas in monophonic, single track artificial composition they appear more plausible). The following table summarizes this section adequately:

	C-RNN-GAN	**MIDINET**	**MuseGAN**	**SeqGAN**
Date (mm/yy)	11/16	3/17	9/17	9/16
GAN Type	GAN	DCGAN	WGAN-GP	Discrete GAN
Solution	3	3	3	1
D Model	LSTM	CNN	CNN	CNN
G Model	LSTM	CNN	CNN	LSTM
Music Representation	MIDI, new format	MIDI, chords	Multi-track Piano-roll	MIDI
Compositional Unit	Note	Bar	Bar	Note
Dataset	hyperlink 1	hyperlink 2	hyperlink 3	hyperlink 4
Curriculum Learning	Partially	No	No	No
Poliphony	Partially	No	Yes	No
Evaluation Metric	Yes	No	Yes	Yes
Feature Matching	Yes	Yes	Yes	No
Open-source code	hyperlink 1	hyperlink 2	hyperlink 3	hyperlink 4
Framework	TensorFlow	Tensorflow	Tensorflow	Tensorflow/ Pytorch

Table 2.1: Overview of previous projects using GANs for musical composition

2.1.3 Other methods applied to music composition

The first and most important project which falls out of range of the two previous sections is WaveNet [35]. This paper presents a novel generative model for raw audio which does not rely on any kind of RNN. Instead, it uses dilated causal convolutions, which are convolutional layers which skip input values at certain steps (for step 1 it uses all the units, for step 2 we use every other unit, etc.) and are stacked on top of each other to produce an output (1 unit) which is based on a very large input, depending on the amount of convolutions and their dilation (step). The output is produced step by step based on the previous time-steps, which resembles the look-back effect of recurrent neural networks , in a way which can only be intuitively understood by analyzing figure 2.4. The problem here is also a reminder of the issue with RNNs: how to consider more previous time-steps when producing the current output. In this case, this problem is formulated as how to increase the receptive field (the amount of input units which participate in the convolutions which lead to the output), which can be done by increasing the dilations and the number of layers, or even by conditioning the network locally on a context stack.

Regardless, this process led to state-of-the-art Text to Speech performance, being limited only by the fact that generation is extremely slow due to the fact that the generation does not allow for much parallelization. Regarding music, the training is made using two relatively generic datasets of continuous audio since WaveNet does not use annotated data. The results are interesting but quite bizarre, featuring compositions that start off slow but ramp up the note density very fast. Even so, these are very impressive results for raw audio, which is not even attempted by most projects in this field, and a large receptive field is said to be crucial for realistic samples, which is not surprising. This architecture received a substantial update in the form of Parallel Wavenet [36], which introduced Probability Density Distillation and a Teacher Student Model, allowing for an effective parallel process which seriously improves generation speed.

Another approach which is unquestionably valuable for music composition is the application of Variational Auto-encoders (VAEs), explored by Sabath et al. [37]. In broad terms, VAEs include an encoder, which compiles all of the data into one encoding that models it in some way. Then we simply sample from this encoding and translate the sample into real generations using a decoder which should mirror the encoder. This leads to a model which is truly generative since we can sample from this encoding without initializing any variables which may bias the composition (such as the priming melodies used in previous projects). This paper uses a specific VAE originally used for image generation entitled DRAW (Deep Recurrent Attentive Writer), which computes the output sequentially, not unlike a traditional RNN. The encoder and decoder are modeled as single-layer LSTM networks, as would be expected. The main limitation here is that the VAE can only produce outputs of the same size as its training data. This leads to two issues: the need to use truncated data chunks and the inability to write long compositions without substantial changes to the model.

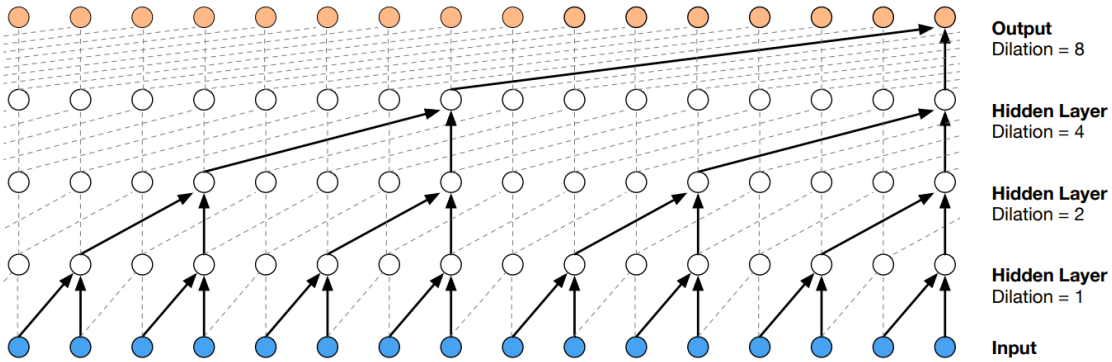


Figure 2.4: The dilated convolutional layers used in WaveNet. The receptive field here is of size 16. [35]

The data used is composed of various Beethoven Piano pieces in the MIDI format, and they are translated to the raw encoding used in most other research papers. The interesting aspect is the objective performance measure introduced, which works by comparing the identity vectors of artificial compositions and real musical pieces. These identity vectors are composed of 17 features meant to characterize the high level sound of each piece which include the number of notes in it and the amount of polyphony throughout its time-steps. Then, one vector can be compared to a Corpus of pieces by using the Mahalanobis distance, given by:

$$D(x, D) = \sqrt{(x - \mu)^T \Sigma^{-1} (x - \mu))} \tag{2.11}$$

This leads to very compelling results, showing that a corpus from a specific distribution is quite close to other corpora of the same kind and quite far from corpora made of randomly generating music. In the end, the generated pieces which attempt to model Beethoven are shown to be quite close to the original corpus, suggesting that this architecture produces realistic results. In essence, the main contributions here are the demonstration of VAEs as music composers and the introduction of an objective, mathematically and musically motivated evaluation measure.

One final tool which is applicable in a wide range contexts including music generation is Curriculum Learning [31]. This technique was mentioned in C-RNN-GAN [3], but its application was very simplified, using chunks of music of increasing size without seriously considering their content. In general terms, the idea behind Curriculum Learning comes from learning at a human level, in the sense that we learn better if we start with the basic and move through increasingly complex concepts and challenges in order to become experts. Although it might seem counterintuitive to compare machines to humans in this regard, this method can be seen as a sort of simulated annealing (which is a staple in Artificial Intelligence) to guide the learning to the global minimum when dealing with non-convex loss functions.

In this specific paper, two promising statements are made and tested using toy ex-

amples: cleaner (less noisy) examples may yield better generalization faster, and introducing gradually more difficult examples speeds-up online training. These would already indicate that curriculum learning is a suitable method, but some more complete experiments are also performed. The first of these trains a neural network to recognize arbitrarily noisy shapes. Starting the training with non-noisy shapes and slowly introducing noise leads to a noticeably lower error rate than just using randomly noisy examples, supporting the first statement. A second experiment is made with a language model that predicts the next word in a sentence, which is easily comparable to music composition. Starting the training with the 5000 most common words in the English Language leads to better long term results than simply sampling from the full vocabulary randomly, supporting the second statement mentioned above. These evidently interesting results, combined with the fact that this work has been the target of various follow-up research papers [38, 39, 40], suggests that curriculum learning could very well be useful in the area of artificial music generation and should be explored for this purpose with more detail.

2.1.4 Deep Q Learning in the context of music generation

In this section we will focus on the RL Tuner architecture introduced by Magenta [2], which we mentioned earlier, and explain its theoretical background.

The foundation of modern Reinforcement Learning is the Markov Decision Process (MDP) [41]. This process defines an environment which can easily describe a large amount of real world problems. This environment is explored by an agent which can perform actions to transition from one state to another at every time-step. According to his state and action, the agent receives a reward. The agent seeks to maximize the total reward he receives at every time-step until the end of the exploration. By definition, an MDP is defined by a vector of 6 elements: $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \gamma, \mathcal{R}^a_{ss'}, \pi \rangle$. We proceed to explain these below:

- The state-space $\mathcal{S}$ - This is a set which contains all of the possible states our agent can be on at any time-step. In the case of RL Tuner the state consists of the LSTM states of the Q-network and the Reward RNN, and also the composition so far. Since the LSTM state of a cell encompasses a continuous range of values, the state space can be described as continuous.[1]

- The action-space $\mathcal{A}$ - This determines the possible actions that the agent can take at any time-step. In our case, this is the note that we will compose next, which can be one of 38 possible values.

- The transition probability $\mathcal{P}^a_{ss'}$ - Given a state and an action, this gives the probability of the next state in the environment. In this case, this concept is not explicitly defined, and the transition is determined by running the note through the reward RNN and the Q network and observing their new states.

[1] However, in the case of RL Tuner, there are only $38^{\text{length of composition.}}$ possible compositions, so the number of possible states is not infinite, even if it is described by continuous values in the LSTM state

- The discount factor γ - This value between 0 and 1 will determine how future rewards will be valued by the agent, which can drastically influence its judgment (the choice of action at each state). Its specific role is mentioned later.
- The reward function $\mathcal{R}^a_{ss'}$ - This will determine the reward that the agent receives at each time-step, depending on his state and action. In our case, the reward depends partially on the Reward RNN and partially on music theory rules which decided the reward based on the action and the previous composition.
- The policy π - This determines the action that should be taken at a certain state. This is what we seek to learn (which note should be played for a specific state of the composition).

After defining the MDP rigorously (as we have done above and in the source code [5]) we can move on to analyze the process of learning a policy. The process applied in this project is, as mentioned earlier, Deep Q Learning, but in order to understand this concept properly we must first define Q values and Q Learning [41, 42].

In short, $Q(s, a)$ represents the reward that will be attained by performing action a while in state s. This is calculated in the following way :

$$(s_t, a_t) = r_t + argmax_a\, Q(s_{t+1}, a_{t+1}) \tag{2.12}$$

where r_t is the immediate reward determined by the reward function. Here we clearly see the role of the discount factor as mentioned before. By learning to estimate the Q values at each time-step we can intuitively learn which action to take at each state in order to maximize the reward. This incremental process is denoted as Q Learning and it is based on exploring the environment and estimating the Q values based on the agent's experiences, updating the Q Values with the following formula:

$$Q(s_t, a_t) \rightarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, at + 1) - Q(s_t, a_t)) \tag{2.13}$$

where α represents the learning rate (the rate at which new experiences influence the Q value). There are also two different ways of learning/exploring: on-policy, in which the exploration is made with the same policy that is being learned, and off policy, where there is a separate exploration policy which is independent of the policy that is being learned. A common alternative to these two extremes is ϵ-greedy exploration, which combines on-policy and random exploration according to the following formula:

$$\pi(s, a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(s)|}, & \text{if } a = a^* = argmax_a\, Q(s, a), \\ \frac{\epsilon}{|A(s)|}, & \text{if } a \neq a^*. \end{cases} \tag{2.14}$$

where a is the action taken, a^* is the greedy action (the action which apparently yields the highest reward) and $|A(s)|$ is the number of actions possible in that specific state.

With the development of Deep Learning and its impressive results with statistical prediction tasks, the neural network clearly appeared as a reasonable way to estimate the Q values. Specifically, Deep Learning was seen as the perfect tool for this procedure since its only limitation is needing a very large amount of data. This is (to a certain extent) not a problem in RL since we can simply create an arbitrarily large dataset by exploring the environment. This led to a paradigm in Reinforcement Learning: Deep Q Learning [43]. This technique was introduced by Google Deepmind and it essentially aimed to perform Q Learning using a deep neural network to estimate the Q Value (know as Deep Q-network), rather than performing the traditional algorithm. This paper introduces a technique known as experience replay, which involves performing exploration (using, in this case, an ϵ-greedy policy) and storing the agents experiences of the form $e_t = (s_t, a_t, r_t, s_{t+1})$ in an experience buffer, which will constitute our dataset. After performing one step of exploration and storing the experience, we sample a random minibatch from our experience buffer and perform a gradient descent step on the following equation:

$$(y_i - Q(\phi_j, a_j; \theta))^2 \tag{2.15}$$

where

$$y_i = \begin{cases} r_j, & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \, max_{a'} Q(\phi_{j+1}, a'; \theta), & \text{for non-terminal } \phi_{j+1}. \end{cases} \tag{2.16}$$

The experience replay method claims two major advantages over standard Q Learning: 1. It can use the same experience twice (experiences are not removed from the buffer during training), which means that the same experience can be used for multiple steps of the gradient descent, which evidently increases data efficiency; and 2. Learning from consecutive samples can be inefficient since they are usually sequentially correlated and this can introduce a bias. Therefore, randomly sampling from the experience buffer allows for a smoother descent and avoids divergence. In the case of RL Tuner, the Q network is a Recurrent Neural Network, which means that the gradient descent is made through the process of backpropagation (which was mentioned in section 2.1.1).

There is, however, one additional technique that is applied in our case, which is called Double Q Learning [44]. To understand why Double Q Learning is needed, it is necessary to understand a fundamental issue with the training process described above. Q estimations are known to be very noisy since the Q Network training process is non-deterministic and sometimes not smooth, and it can start fitting the noise in our dataset. This means that when we calculate y_j (as described above), the Q estimates will be noisy. However, since we want to maximize the Q value, it is likely that the Q Value with most upward noise will be chosen, and will then be used to update the Q Network. This means that our Network is trained to match a potentially overestimated value, and since the Q Network in 2.15 and 2.16 is the same, this overestimation will propagate and reach unrealistic values. This would be acceptable if all Q values were overestimated equally, but since they overestimate according to

random noise, this can lead to some Q values being much more overestimated than others, making our Q estimation noisy and unreliable. This can obviously influence our finally policy and yield sub-optimal results. However, this issue can be solved by using two separate networks: the Q network for 2.16 and the Target Q network for 2.15. This solves the issue of propagating the overestimations and can therefore improve results, as was experimentally demonstrated in [44]. The two networks can both be kept up-to-date by symmetrically updating them (switching their parameters often) or by using a slow moving copy of the Target Q network as the Q network (the latter is used in the RL Tuner). The target for Double Q-learning is therefore described as:

$$y_i = \begin{cases} r_j, & \text{for terminal } \phi_{j+1} \\ r_j + \gamma\, Q(\phi_{j+1}, max_{a'} Q(\phi_{j+1}, a'; \theta_t); \theta'_t), & \text{for non-terminal } \phi_{j+1}. \end{cases} \tag{2.17}$$

In the case of RL Tuner, both the Q Network and the Target Q Network are LSTM networks. This means that the state which is used as input is the LSTM state only. This would seem contradictory to how the state was described when we defined our MDP, but we must keep in mind that this LSTM state should be able to convey the information included in the Reward RNN LSTM state and in the previous composition, and therefore learning the Q values is entirely possible and fair for the Q network.

2.2 Overview of key music theory concepts

Since a considerable amount of music theory knowledge is necessary to fully understand the intentions of this project, it is evident that some simplified background concepts must be explained. The following information was mainly gathered from the following sources: [45, 46]

2.2.1 Fundamentals of music theory

A musical composition can be seen as a sequence of notes and pauses. Notes are sounds played at a certain time-step with a certain duration and a certain pitch, while pauses (or rests) represent the absence of notes, so they are conceptually similar to notes except for the fact that they do not possess pitch. If we consider the piano-roll format, we can think of a musical composition as a 2-dimensional matrix. Pitch represents the frequency of a sound (intuitively, how high or low it sounds) and it is seen below as how low or high the note is in its column. The time-step in which a note occurs can be called its time and it is seen below as the position of the note in its row. The duration of the note defines how long it sounds and it can be seen as how much horizontal space it occupies in its row (or how many squares it occupies).

The pitch of a note could be seen as continuous (the frequency of a sound is a continuous value), but it is commonly divided into a specific set of possible pitches which increase exponentially in frequency and are represented as the rows of the

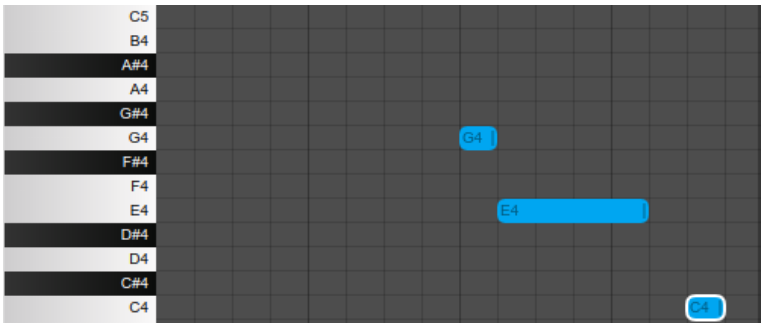


Figure 2.5: A very simple melody represented in the piano-roll format

matrix. This set is then divided into groups of 12 pitches, which are called octaves. In figure 2.5 we can see one full octave, which goes from the first note C4 to the last note C5. C4 represents the relative pitch (C) in the context of the 4th octave. We can play a note in any of these 12 possible pitches.

Each of the columns can be seen as a beat where a note or pause could be played. These beats are then grouped into bars, depending on the meter. For instance, if the meter is X/4, each bar will contain X quarter notes (or pauses), X*2 eight notes or X/2 half notes. The duration of one single column (or square) is defined a priori, and we will further on consider it to be one sixteenth note. The meter used will be 6/8, meaning one bar will contain 12 sixteenth notes (12 columns in piano-roll).

If we consider a sequence of notes and pauses without taking into account their pitches, we are analyzing the rhythm of the piece. If we also take into account the pitches and the piece is monophonic (only one note is allowed in each column), we consider it a melody. Harmony, texture and timbre are also fundamental music theory concepts, but it is unnecessary to define them since they are not relevant for this project. Note that we will only be considering simple discrete representations of music, whereas traditional music sheet representation and raw audio have other complexities.

2.2.2 Fundamentals of melodies

Now that we have understood the nature of musical notes and how they form melodies, we can explore some more intricate concepts. As previously mentioned, one octave contains 12 notes, which are separated by one semitone each, forming the sequence C C# D D# E F G G# A A# B C. Note that E# designates the same note as F. If instead of taking all of the notes we consider them using the following sequence of distances: tone (equivalent to two semitones) tone semitone tone tone tone semitone, we get a major scale: C D E F G A B C (in this case, C major scale, since it starts with the note C). This is an organization of these notes in a way that makes them sound adequate in direct or indirect sequence. The first note of the scale is know as

the tonic. There are many different types of scales, but the ones which are relevant for this project are the major scale and the minor scale. The minor scale is defined by the following sequence of distances: tone,semitone,tone,tone,semitone,tone,tone.

As stated previously, a scale only spans one octave. However, if we consider the C Major scale in every octave, we are considering every note in the key of C Major. A piece can be described as being in the key of C Major if its notes are (mostly) contained in this set of notes. It is worth noting that the key of C Major also contains the notes of every A minor scale, and therefore the key of C Major and the Key of A minor can be seen as the same set of notes, although musical pieces in these two keys can differ somewhat due to the choice of notes from this set. Formally, A minor is described as a mode of C Major, which means that it uses the same notes as the C Major Scale but starts from a different note (a different tonic). There are evidently other modes but for the context of this project this topic is not worth exploring in depth.

Once we grasp the concept of a scale and key, we can start understanding intervals. An interval is the distance between two notes in a key. For instance, if we are in the key of C and we play the note C followed by the note D, we have a major second, which represents the distance between two consecutive notes in this key separated by one tone. If we played E followed by F, we would have a minor second, which represents the distance between two consecutive notes in this key separated by one semitone. The denomination of intervals within the context of a scale or key is defined according to the following legend: One final concept which is essential for

Length in semitones	**Possible intervals**
0	Perfect unison
1	Minor second, augmented unison
2	Major second, diminished third
3	Minor third, augmented second
4	Major third, diminished fourth
5	Perfect fourth, augmented third
6	Diminished fifth, augmented fourth
7	Perfect fifth, diminished sixth
8	Minor sixth, augmented fifth
9	Major sixth, diminished seventh
10	Minor seventh, augmented sixth
11	Major seventh, diminished octave
12	Perfect octave, augmented seventh

Table 2.2: Complete description of all possible intervals according to music theory.

understanding this project is relativity in music. If we analyze our two fundamental musical concepts - pitch and time - we will see that any melody is relative with respect to these. For instance, a certain melody can start after an initial pause and still

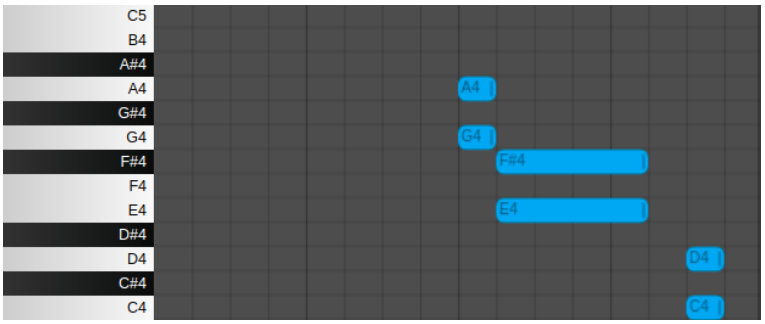


Figure 2.6: A very simple example of transposition. Here we see the melody displayed in 2.6 played simultaneously with a transposed version of this melody (two semitones above the original). We can see that the intervals and the rhythm remain equal between the two melodies.

evidently sound the same, meaning that the melody can be placed freely along the time axis and still sound the same, as long as the duration and relative time-steps of its notes and pauses are the same. We can also understand that the real duration (in milliseconds) of the sixteenth note (one square in our piano-roll representation) is not defined, which means that this as well is relative. This depends on the tempo of the piece, which can be defined in beats per minute (bpm). This will not be considered in this project, but it is useful to keep in mind that any melody represented in the piano-roll format can be played slower or faster without losing its characteristics. The final element of relativity is, of course, pitch. This is a difficult concept to grasp without listening to the correspondent audio samples, but if all of the notes in figure 2.5 were shifted one unit (semitone) upwards with respect to pitch, we would have a melody that would sound the same, but slightly higher. This means that when we shift every note in a melody with respect to pitch (this process is typically called transposition and is displayed in figure 2.6) by X semitones, we completely change the sequence but the relative distance in pitches, duration and time stay the same, which means that the melody retains its features and musicality. The concept of transposition will be extremely important for one of the main contributions of this project.

Chapter 3

Dataset

In this chapter we will describe the dataset which we used to train our model. We begin with a general description of the dataset, explaining the genre which it pertains to and the reasons for choosing it. We then move on to explain the formats that were used while gathering the dataset (MIDI and Guitar Pro 5), and how they differ from continuous musical representations such as raw audio. We end by illustrating how we collected these files and converted them to a monophonic discrete format.

3.1 General description

The choice of dataset is central to any project in Machine Learning, and artificial music composition is no exception. Many of the existing projects using recurrent neural networks have used either popular music [17], classical music [19] or simply a general compilation of pieces from various different contexts [18]. However, for this project, we decided to gather a more conservative yet uncommon dataset, which was made entirely of pieces within the genre of *Xota Gallega*, a variation of the Spanish folk music *Jota* which is practiced in Galicia. The most common melodic instruments in this genre include the bagpipe and the guitar. The reasons for this choice were numerous. The first and most important reason was the simplicity and consistency of the compositions. Pop music, for instance, suffers from extreme variance between different pieces and can have subtleties and complexities which are hard to model and describe. Xota does not have this issue since its pieces usually have similar rhythmic patterns and pitch variations which are repeated throughout the song. This is helpful in order to manually extract the compositional rules which the pieces generally adhere to, as we will mention later. Another surprising advantage to this dataset was the availability of real compositions in discrete musical formats, namely MIDI and Guitar Pro 5 (GP5). This made it possible to gather a very consistent and somewhat broad set of MIDI files which could be used to train our RNN.

3.2 Musical formats

There are many possible discrete representations of music. The most well known of these is of course sheet music. However, this is not a computational format and its translation to numerical form is not necessarily direct. Therefore, other formats were developed in order to represent music discretely in numerical, computable terms. The most common of these is MIDI [47]. In short, MIDI encompasses the information present in the piano-roll format mentioned in 2.2. It contains the time, pitch and duration of every note, each of these represented by a discrete value. It also contains the tempo of the piece (in bpm) and the instrument that plays it, which is one of a limited set of possible instruments with pre-programmed timbres. For this reason, it is not reasonable to say that MIDI models the timbre of the composition given the limited number of choices, whereas raw audio can express this in a continuous form. MIDI files can also contain multiple tracks, representing the various melodic lines that are played in parallel during the piece. MIDI files have many other important characteristics, but this simple description will suffice for the context of this project.

Guitar Pro 5 (GP5) is another digital format which represents essentially the exact same information as MIDI, as well as some technical information about manually playing the piece, and it is generally used in the context of learning how to play the pieces on the guitar. *Crucially, both of these formats can model polyphonic music (multiples notes sounding simultaneously), which is what separates them from the representation that the RL Tuner uses* . This means that a conversion between these formats is required and not necessarily obvious, given that they differ in this fundamental aspect.

3.3 Collection process and preprocessing

Number of MIDIs	Avg. size of MIDI (Kb)	avg. num. of events in MIDI	Total size of TF Records (Mb)
261	2.96	859.97	46.8

Table 3.1: This table displays relevant statistics about the content of our dataset.

The first step in gathering the dataset was, of course, gathering the music files from various sources [1], which were found by Emilia Parada-Cabaleiro. After this, it was necessary to convert some of the files to MIDI, since they were in the GP5 format. This turned out to be a substantial issue since Guitar Pro 5 is a patented product and its official software is not open-source. This means there is no obvious way of

[1]The sources used were the following: http://www.folkotecagalega.com/pezas/jotas, http://perso.wanadoo.es/marco.velez/repertor/busca/ritmo.htm, http://www.gaitagallega.es/repertorio-de-gaita/jotas.html, https://partiturasgaitagalega.wordpress.com/partituras/partituras-en-do/ .

converting Guitar Pro 5 files to MIDI in batch, which is extremely problematic. Fortunately, there is a publicly available website [48] which performs this conversion, even though not in batch. Therefore, we developed a script using Selenium [49] to convert the files in batch without having to access the website manually multiple times. This worked well and is one of the contributions of this project, since gp5 files are quite common and converting them to MIDI could prove useful in other similar projects.

After gathering the MIDIs, it was apparent that most of these contained various simultaneous tracks, which meant the compositions were polyphonic. As mentioned previously, RL Tuner cannot deal with polyphony, so the pieces had to be separated into different MIDI files for each individual track in each composition. Since for most polyphonic compositions the second (and possibly third) track consisted roughly of the same melody as the first but transposed upwards or downwards, only the first track of every MIDI file was kept. The separation was made using the python library MIDO [50]. In the end, our MIDI dataset was comprised of 261 *.mid* files, totaling 773.6 kB. After this process, all that is left is creating note sequences (not unlike the format used by RL Tuner) from the MIDI files and then converting these note sequences into sequence examples which can be used directly to train and evaluate the Melody RNN. These two transformations were made using the open-source code provided by Google Magenta to support their framework.

Chapter 4

Melody RNN

In this chapter we will speak about the Melody RNN as a generative model for music and as a tool to produce checkpoints which will be loaded by the RL Tuner. We begin by briefly describing this model and its three different variants. We proceed to explain the training procedure in more detail, explaining the meaning behind all of the parameters and how they will affect the training process, as well as the values to which they were during the experimental phase. The next part of this chapter is divided in two sections: Attention Mechanisms and Overfitting. The first of these explains how we implemented attention mechanisms for the Basic RNN in order to produce checkpoints that could be used with the RL Tuner, which is one of our contributions, and explains why this couldn't be done with the Attention RNN. The second discusses the issue of overfitting, detailing why this is especially problematic for this application and how it can be alleviated by monitoring the training accuracy and augmenting the dataset. We finish by explicitly stating the RNN configurations we will be using for the main experiments.

4.1 Brief description of the model

The Melody RNN [51] is a model developed by Google Magenta intended for composing discrete artificial music (MIDI files). It consists of an LSTM network which can be adapted by providing different parameters, similarly to what was mentioned in 2.1.1. The training is performed using MIDI files, which are converted to TF records that the network can take as input. The Melody RNN has three different variants:

- **Basic RNN** - a regular LSTM network which uses an encoding (for the musical pieces) which is very similar to the one used in RL Tuner.
- **Lookback RNN** - a regular LSTM network which uses a much more complex lookback encoding (mentioned in 2.1.1).
- **Attention RNN** - an LSTM network which uses attention mechanisms (specifically, an attention wrapper on the LSTM cell which implements these mechanisms) and **also uses the lookback encoding** (mentioned in 2.1.1)

Characteristics	Basic RNN	Lookback RNN	Attention RNN
LSTM network	✓	✓	✓
Uses RL Tuner encoding	✓		
Uses lookback encoding		✓	✓
Uses attention mechanisms			✓

Table 4.1: This table explains the three different Melody RNN configurations, detailing their similarities and differences.

4.2 Overview of the basic training procedure

The first of our training attempts was done with the simplest of the three variants: the Basic RNN. Special care was taken to make our training of this model as similar as possible to the training process that had generated the pre-trained (with thousands of hours of arbitrarily gathered music) checkpoint published by Magenta. This was done by carefully analyzing the checkpoint variables and their dimensions. The parameters for this model were the following:

- **Batch size** - This determines the size of the batch which will be fed to the model in each training step. It was originally set to 128 and was kept at this value since it matched the pre-trained checkpoint. This yielded satisfying results and there was no need to lower it since we did not have substantial memory constraints.

- **RNN Layer Sizes** - This determines the number of LSTM layers we will use and the size of each layer. It was originally set to $[128, 128]$, which means that we would be using two layers with 128 units each. However, this was changed to $[512, 512]$ in order to match the pre-trained checkpoint. Theoretically, this means that the network we trained is able to model more features than the default model.

- **Dropout keep probability** - This determines the probability of units being kept during each training step, and it is typically used to reduce overfitting. The value of this parameter is equal to $1 - d$ where d is the dropout probability. It was originally set to 0.50, which is already quite low. This is an important parameter which will lead to further discussion in 4.4, and it was kept at its original value.

- **Gradient clipping norm** - During training, the RNN may suffer from a common problem mentioned earlier, known as Exploding Gradient, which can lead to NaN (Not a Number - a number too small or large to be represented digitally) gradients and interrupt the learning process. To partially alleviate this issue, the gradients' norms are capped (clipped) at a certain value in order to prevent them from getting too large. This parameter controls this cap and was

originally set to 5. This value led to an apparently smooth training process and was the same as the pre-trained checkpoint, so it was left unchanged.

- **Learning rate** - This determines the learning rate used in the Adam optimizer [22], which is used to update the network weights. It was originally set to 0.001, which is standard, as proposed in [22]. A higher value could have sped up the initial training phase but this is not necessarily desirable. Since the pre-trained checkpoint also used this value, the parameter was kept at 0.001.

The number of training steps is also an important part of this process, and these will be discussed thoroughly in the section 4.4. During training, the model regularly saves checkpoints at different time-steps and records the models' accuracy with the training data (among other measures) at that specific point in training.

4.3 Attention Mechanisms

The initial idea behind this project was to combine two Magenta models: the RL Tuner and the Attention RNN (a variant of the Melody RNN). This was mainly due to the fact that these two models appeared to have the most impressive samples [2, 6] without using any shortcuts or dubious methods. This combination appeared to involve a simple modification of the RL Tuner to accommodate loading a checkpoint given by a regular LSTM (namely the Basic RNN) with additional attention mechanisms. This assumption was due to the fact that in [6], the Lookback RNN and the Attention RNN are shown as two vastly different variants of the Melody RNN: the Lookback RNN relies on a new, more complicated encoding for the input of the LSTM, which adds features such as a binary value to indicate whether the previous note was the same as in the last two bars or not; while the Attention RNN uses attention mechanisms to improve the outputs. This is not entirely untrue, but the fact is that after analyzing the code, we found that the Attention RNN actually builds on what is presented with the Lookback RNN: it uses attention mechanism *as well as the lookback encoding*, as seen in figure A.1 (see appendix).

This aspect, even if it appears somewhat minor, is crucial since it makes the Attention RNN incompatible with the RL Tuner due to their different encodings. This incompatibility is not easily solvable since the entire RL Tuner architecture relies on its encoding and using a checkpoint with a different encoding would necessarily involve restructuring the code in various ways. However, since we were only interested in the attention mechanisms, we took a different approach to this issue. We decided to modify the original Basic RNN so as to imbue it with attention mechanisms, so that we would then produce a checkpoint which would be compatible with the standard encoding while still having the attention mechanisms. This involved some changes to the Melody RNN code in order to make the Basic RNN's LSTM cell have the Attention Wrapper. This worked well, and after some changes to the RL Tuner code, it was compatible with our new Attention RNN (which we will denote as New Attention RNN for the remainder of this report). Specifically, the new model was trained with the same parameters as the Basic RNN apart from the following:

- **RNN Layer sizes** - For this parameter, we could not use the value of $[512, 512]$, due to the fact that it quickly led to a NaN loss. This is probably due to the fact that the increase in units led to vanishing/exploding gradients, but further experimentation with other parameters could not solve this issue, so this parameter was changed to $[128, 128]$.

- **Gradient clipping norm** - As mentioned in the previous point, exploding gradients are still a very large issue. Although attention mechanisms provide a better way to take into account the past input, they still suffer from numerical instability. Therefore, the clipping norm was changed to a more restrictive value - 3 - as it is in the original Attention RNN.

- **Attention Length** - This parameter controls how much of the previous input we should consider for the attention mechanism. In this case, we chose to consider the 40 previous events since this was the value of this parameter in the original Attention RNN and it led to a smooth training procedure.

4.4 Overfitting in Artficial Music Composition

Overfitting is a very well known issue amongst the Machine Learning community. Essentially, a predictive model overfits when it fails to generalize the dataset and instead fits its noise, creating a model which displays no real intelligence and generally has no value. However, in discrete creative fields such as music composition with MIDI, the data does not typically contain noise, as opposed to raw audio. This means that the overfitting issue is usually expressed in the form of replicating chunks of the dataset instead of composing original pieces. This problem is not often mentioned in academic papers (since it yields poor results) but is informally very well know to researchers in this area using RNNs. For these reasons, properly dealing with overfitting was an expected requirement in order to successfully develop this project.

Our first attempts at training the RNNs used the parameters mentioned above and ran for 20000 training step, as suggested in [51]. This led to some fairly pleasing results. However, after observing the note probability density graphs produced by the original RL Tuner code, we noticed severe overfitting. This was especially the case for the Basic RNN, but it was also apparent for the New Attention RNN, as we can see in figure 4.1, compared to the pre-trained Basic RNN. The pre-trained RNN most likely did not overfit due to its very large and broad dataset. Our dataset is limited, which makes it relatively easy for the network to memorize the pieces instead of generalizing their style. As it can be seen, attention mechanisms do not help with this issue, although the Attention RNN ends training with a slightly lower accuracy since it generally requires more training steps, as we can see in figure B.5 (see appendix). Increasing the dropout is also not a viable option since it is already quite high (0.5) and making it higher would disrupt the learning process too harshly. After observing the accuracy of the model during training, it is quite apparent that there is an excess of training steps, since after a certain point the accuracy becomes so

close to 1 that the model is clearly overfitting. The conclusion which can be drawn here is that the overfitting is caused by two major factors: an excess of training steps and a lack of sufficient data. The first issue can be solved by monitoring the training accuracy (as seen in Figure 4.2) and will be addressed during Chapter 6 by trying different amounts of training steps, but the lack of data can be alleviated by a more remarkable solution.

Data augmentation is a relatively well known tool when using Deep Learning to deal with limited datasets. It is, for instance, very useful in the context of image classification [52]. Let's assume we are trying to recognize a cat in a picture by using a Convolutional Neural Network to classify it as containing a cat or not. We feed it a cat picture during training. If the cat is on the right side of the picture, the network detects that this specific pattern on the left of the picture generally means that it contains a cat. Let's say we now feed it the same picture, but mirrored. As a human, one would be inclined to say that the network is not going to learn anything new since the content of the picture is still the same, i.e., it does not constitute new data. However, now the network learns that this mirrored pattern on the left of the image means that it contains a cat, which is not obvious for the network since it has no way of recognizing the mirrored picture. This means that by mirroring every image we have we can duplicate the size of our dataset without directly adding redundancy to the learning process. This concept is know as Data Augmentation and is especially useful since Deep Learning methods usually require very large datasets. For music, this process is not obviously applicable, but if we take into account the relativity of music through transposition, as mentioned in 2.2.2, we realize that the augmentation of a musical dataset is actually feasible.

The idea behind the augmentation is to simply take every piece and transpose it to every tone possible in the octave below and above it (shift all of its notes by X semitones), as it is proposed in other works using Raw audio [53] and MIDI [7]. When we apply this transformation to a piece, its musicality stays the same, and as in the previous example, one might be tempted to assume that this new data is redundant. However, if we train the LSTM with a simple piece where the note D is followed by note E, it learns that, for the input correspondent to D, the ideal output should give high probability to E as the next note. However, doing the exact same process with D# and F (the previous melody transpose one semitone up) trains the network with a different input and a new output, while still dealing with the same interval (major second) and therefore retaining the piece's sonority.

This process creates 24 pieces for every single piece in the dataset, ideally multiplying our amount of data by 25. Evidently this created some pieces with notes which were out of the range allowed according to the conversion between MIDI and sequential encoding performed by the Melody RNN, but the Magenta code is equipped with measures to discard these pieces, so this did not pose a major issue. The transposition was attempted using various python libraries but these methods generally changed the compositions slightly, so we decided to do this simply by converting the

MIDIs to the sequential encoding and transposing them by adding a certain value (-12 to 12) to every note of the sequence. This method worked quite well and the training melodies went from a mere size of 26.9 MB to a much more substantial 473.7 MB.

It is clear by looking at figures B.1 and B.2 (see appendix) that augmenting the dataset increases the number of relevant training steps before saturating the accuracy and subsequent overfitting, which is in itself a demonstration of how this method can be beneficial towards any Deep Learning model trained with MIDI files.

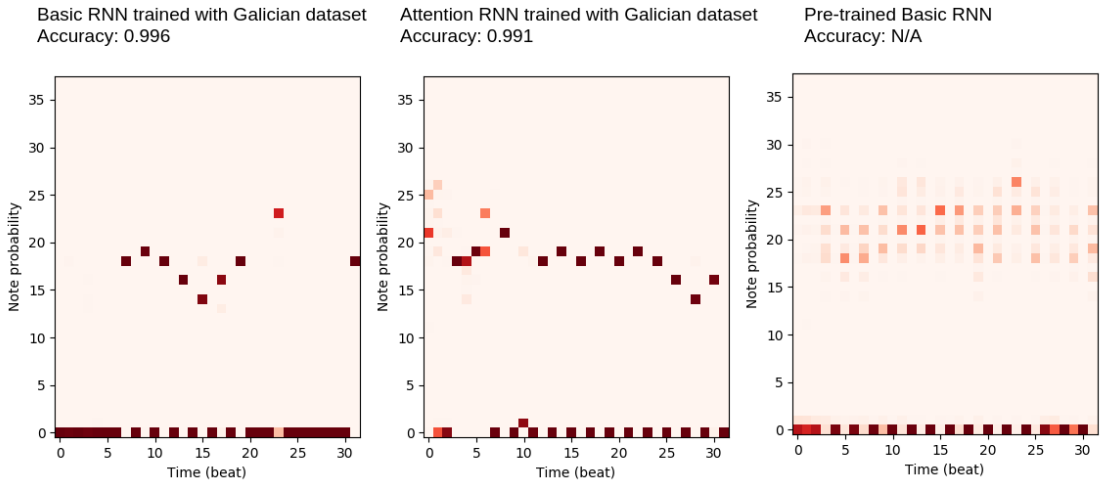


Figure 4.1: Due to the Pre-trained RNN's broad dataset, its note probability density is very well distributed, while the other networks which are clearly overfitting have very narrow probability densities.

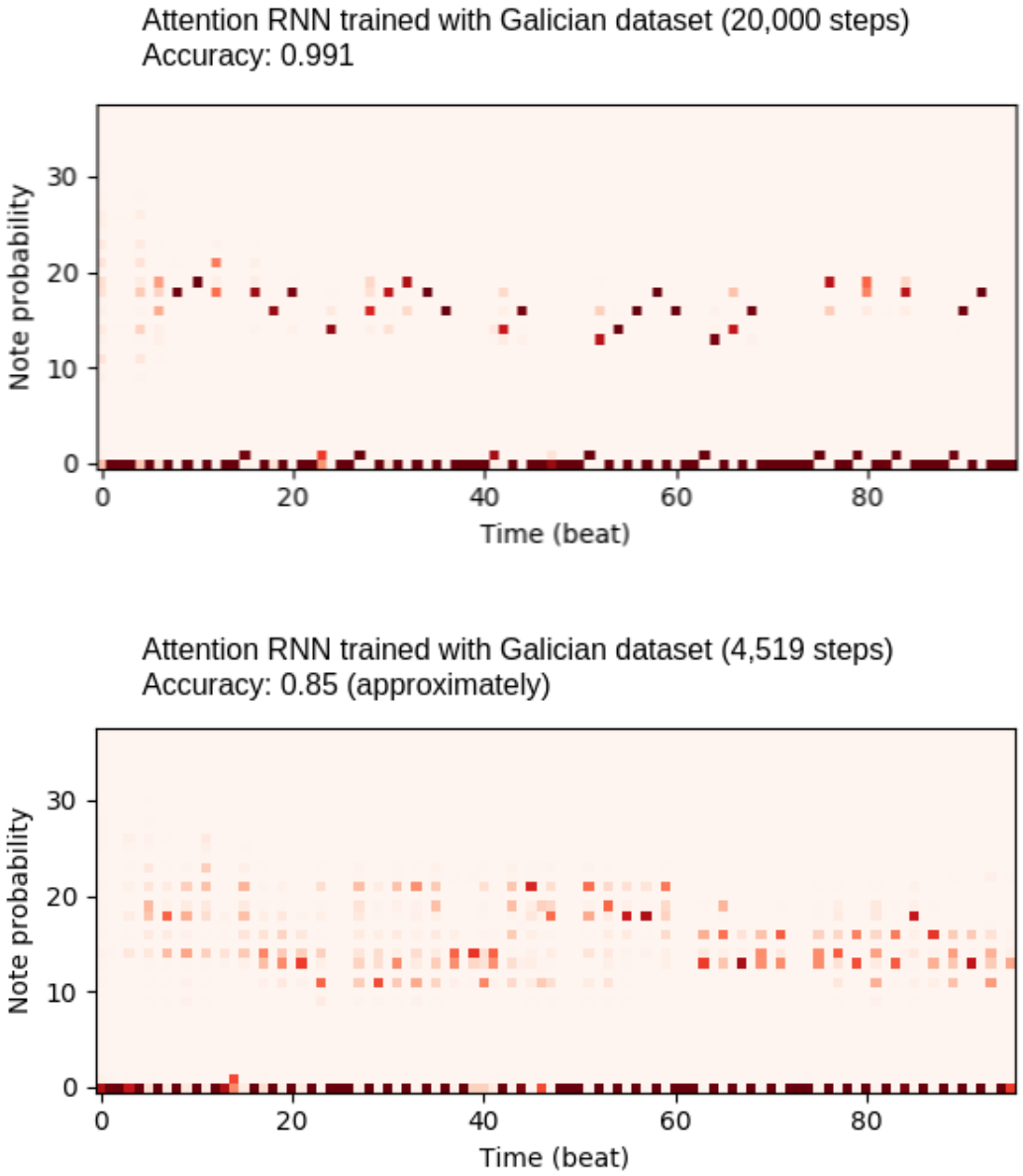


Figure 4.2: Here we can see how monitoring the training accuracy of a network can be very important. We can see that the first network is very confident about the notes due to its high accuracy, which will cause problems with the RL as we will describe later. The second network, on the other hand, is much more flexible and has various different possible notes at every time-step.

4.5 Final configurations

After the initial experiments and the work detailed in the sections above, we focused on four different Melody RNN configurations which will be used for the experiments in chapter 6. The Pre-trained Basic RNN is simply the Basic RNN loaded with the checkpoint taken from Magenta [51] (trained with thousands of MIDI files). The Galician Basic RNN is similarly the Basic RNN trained with our Galician dataset. The Pre-trained+Galician Basic RNN is the Pre-trained Basic RNN trained with the Galician dataset after loading the pre-trained checkpoint. Finally, the Galician Attention RNN is the New Attention RNN (as mentioned earlier in this chapter) trained with the Galician dataset. The three last RNNs were trained by us and were monitored (seen in figures B.3, B.4 and B.5 - see appendix) to produce checkpoints at different training accuracies:85, 90 and 95 %, to evaluate the effect of overconfidence/overfitting when using the RNNs with the RL Tuner (mentioned below). All of this information is summarized in the table below:

Characteristics	**Pre-trained Basic RNN**	**Galician Basic RNN**	**Pre-trained + Galician Basic RNN**	**Galician Attention RNN**
Uses RL Tuner encoding	✓	✓	✓	✓
Based on the Basic RNN model	✓	✓	✓	✓
Uses attention mechanisms				✓
Trained with Magenta dataset	✓		✓	
Trained with Galician dataset		✓	✓	✓
Trained for different training accuracies		✓(85,90,95 %)	✓(85,90,95 %)	✓(85,90,95 %)

Table 4.2: This table explains the four different Melody RNN configurations which will be referenced for the remainder of this report, explaining their similarities and differences.

Chapter 5

RL Tuner

In this chapter we will describe the RL Tuner training procedure, as well as the original and new rule-sets which we will use for the experiments. We begin by characterizing the model itself, referring to previous sections such as the background, and specifying the musical representation used in the RL Tuner. We then proceed to analyze the original rule-set and present the new rule-set, specifying the rules which are contained in them, the behaviors they seek to encourage and the reward functions which implement them. We also mention the combination of these two rule-sets, and explain why it is also relevant for the experimental phase. We then provide the configuration details, explaining the parameters, their meaning and the value to which they were set, as we did in the previous chapter. In the end, we describe some implementation details about the modifications and extensions which were crucial in order to make the RL Tuner fit our needs for this project.

5.1 Brief description of the model

The RL Tuner has already been mentioned in sections 2.1.1 and 2.1.4, but it is important to define it as a model before we extend its functionality. The RL Tuner consists of three LSTM networks. One of represents the Deep Q Network; the other represents the Target Q Network (since we are using Double Q Learning, as mentioned in 2.1.4); and the third represents the Reward RNN. The first and second networks play traditional roles explained in 2.1.4, while the Reward RNN provides part of the reward for the MDP. The other part of the reward is given by a music theory reward, which can be programmed by the user and can depend on the current action and the previous composition. The networks are all initialized with a checkpoint given by a trained RNN (in this project, we will be using the Basic RNN mentioned in the previous chapter). As the network is trained, the Reward RNN remains fixed while the Deep Q Network is updated using the traditional Deep Q Learning algorithm (mentioned in 2.1.4) and the Target Q Network is gradually updated to resemble the Deep Q Network. In the end, the Deep Q Network is used to compose as a traditional LSTM network (using priming melodies as mentioned in 2.1.1).

Before moving on to the rule-sets, it is crucial to explore the RL Tuner representation

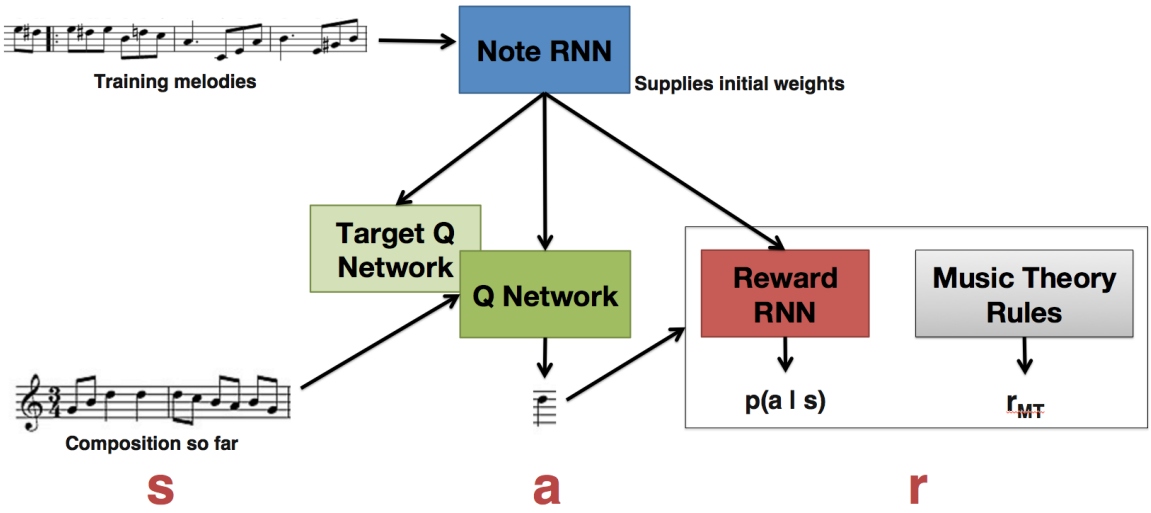


Figure 5.1: The Reinforcement Learning Tuner architecture, where *s*, *a* and *r* represent state, action and reward respectively. [11]

(or encoding) in detail. The RL Tuner considers pieces as one-dimensional vectors. Its format is therefore a collapsed version of the piano-roll format mentioned earlier, in the sense that the pitch is represented as a numerical value instead of a position in a column, while the position in the vector still represents the time-step in which the note occurs. In total, 36 possible pitches (spanning 3 octaves) are allowed and represented by the integers 2-37. The remaining integers 0 and 1 are denoted as special events. The first of these is the held note event (0), which represents the holding of a note or pause. The second is the note-off event, which represents ceasing to play note, or beginning a pause. This means that if our unit is the sixteenth note, a quarter note would be represented as [X,0,0,0], where $X \geq 2$, while a quarter pause would be represented as [1,0,0,0]. It is worth noting that consecutive zeros in the beginning of the composition represent a pause since no value greater or equal than two has appeared. Evidently, this format can only represent monophonic music, which means that the RL Tuner must be trained with and compose pieces with only one note/event per time-step.

5.2 Description of the different rule-sets

As mentioned previously, part of the reward given for an action in our MDP is given by the programmer. One can decide to give a reward based on the action (the next note) and the previous composition. Essentially any pattern can be theoretically programmed to be rewarded, but this does not guarantee any impact in the post-RL compositions. This *music theory reward* is then determined by what we will denote as a *rule-set* which determines what kind of behavior should be rewarded/punished. Below, we will explore the various rules-sets which were experimented with. All of the statements made below about the effectiveness of each rule is justified by the results in the project repository [5] and in chapter 6.

5.2.1 The original rule-set

The intention of the original rule-set [2, 54] is to tune compositions in a non-specific style, simply correcting the frequent issues of pre-trained RNN compositions such as extreme repetition or auto-correlation, and adding some markers which are very common in almost all western music, such as starting the composition with the tonic note or remaining in the same key throughout the composition. The rule-set is described as such:

1. **Stay in key/Follow the scale** - The concept of staying in the same key is generally followed in traditional musical composition and helps give the piece a more consistently pleasing sound. In this case, it is realized through three reward functions: one rewards the composition for being in key, the other punishes it for being off key and the third rewards notes in the scale. Our compositional range here, as mentioned previously, spans 3 octaves and therefore the keys contain any appropriate notes in that range. The scales which are rewarded here, on the other hand, encompass only the full scales, from tonic to tonic, that can be played in this range. Therefore, the key of C Major, for instance, is composed of the following notes [0, 1, 2, 4, 6, 7, 9, 11, 13, 14, 16, 18, 19, 21, 23, 25, 26, 28, 30, 31, 33, 35, 37], whereas the scales only contain the following [2, 4, 6, 7, 9, 11, 13, 14, 16, 18, 19, 21, 23, 25, 26], where [2,14,26] are the tonics. In the rule-set which is actually applied, only the second reward function is used. It is worth noting that if no scale is selected, the model will automatically reward C Major and support for other scales is very minimal, having required some additions to work adequately.

2. **Start and end with the tonic tone** - This idea is also very common in most popular music. However, the way in which it is implemented leaves a lot to be desired. The composition is rewarded if and only if the tonic note is the action and there are no previous notes (the first note of the composition will be the tonic) or if the tonic is played four notes before the end of the composition. Both of these behaviors are not necessarily desirable. Forcing the composition to start with the tonic note instead of allowing it to start with a pause and then compose the tonic note does not make sense and assuming that the bar length used is four is also not adequate for every meter. However, the first of these rewards was quite successful with the original model (in the sense that it heavily impacted post-RL compositions), which is an important achievement.

3. **Avoid excessively repeated notes** - Repetition of the same note multiple times in a row can easily remove all musicality from a piece, and should therefore be avoided in largely any musical composition. This rule is applied in terms of a reward function which penalizes the composition for writing more than eight repeated notes in a row with pauses and held notes in between, more than six repeated notes with pauses *or* held notes in between, or more than four repeated notes with *no* pauses or held notes in between. This tolerance for repetition with some rhythmic variety is reasonable and the punishment expressed in reward form is extremely severe, leading to compositions which generally boast a very low percentage of this type of repetition.

4. **Prefer harmonious intervals** - As mentioned before, the main elements which define a melody are its rhythm and its intervals, everything else is relative. Therefore, in order to compose pleasing music, it is very important to encourage intervals which generally sound good and punish dissonant ones. In the code, this is done using a relatively complex function (which had to be manually changed in order to accommodate keys other than C Major) which detects the interval between the last note played and the next note (the action) and whether it is in key (whether both of its notes are contained in the key we are composing in). Depending on this classification, another reward function rewards common intervals such as perfect fifths or major thirds in key, and punishes intervals such as sevenths or eighths since they are too wide and generally sound strange. In the end, this is a flawed method since it is hard-coded for the key of C major only and makes plenty of bold assumptions, but it works decently well. In the implementation of this reward, there are 'preferred' intervals, which in C Major are the major third C - E, and the perfect fith C - G.

5. **Resolve large leaps** - This is perhaps the most generally applicable of all the rules. In almost all compositions, the notes during a small chunk of time are generally contained within a somewhat narrow tonal range. This means that the composition moves slowly and the notes are generally consistent with each other, in the sense that the height of their pitches is similar. Therefore, when a large leap occurs (i.e., a large interval), the composition usually leaps back to the original tonal range within the next notes. This is done to give the song some variation and keep the listener engaged while still keeping track of its original place in the pitch spectrum. This is applied in a large variety of genres and is a staple of western music. In the code, this is implemented quite elegantly through a function which detects the leap (an interval larger or equal to a perfect fifth) and its direction, and then rewards the composition if it detects a similar leap in the opposite direction within the next six notes. This is an extremely well realized rule that successfully promotes tonal consistency, which is one of the most difficult tasks in music composition.

6. **Avoid repeating extrema note** - This rule serves roughly the same purpose of the previous rule. The concept of keeping a narrow tonal range is very important, and a good way of asserting this is to reward the composer for using the highest and lowest notes of its range only once. This steers the composition towards the middle of the tonal range instead of its edges, which in a way enforces the tonal range. This is done by a very simple function which, after the last note is composed, rewards the piece for having a unique low note and a unique high note. This method of rewarding is relatively sparse due to the fact that the reward is given at the end of the composition process but is clearly dependent on the whole process, and also because the composer is not rewarded for having the minimum/maximum note appear only twice. Nevertheless, the reward appears to be quite effective.

7. **Avoid high auto-correlation** - This method is easily comparable to rule 3 since

it serves the same principle: avoid noticeable repetition. Although repetition of patterns is an integral part of modern music, this repetition must be tasteful. In order words, the pattern which is repeated must have some variation if it is repeated often, and even then it is generally unwise to repeat it throughout the whole song if we are composing the lead melody. The idea here is to punish the composer for composing short patterns which are very similar to the ones it just composed. In practice, this is done by calculating the correlation between the current composition and the previous composition with a lag of 1, 2 and 3 using the function shown in figure A.2. If any of these are relatively high, a negative reward is emitted.

8. **Play motifs** - Motifs are generally considered to be the core of any complex melody. In short, the motif represents a short but generally memorable sequence of notes, which is then repeated, often with slight variation, throughout the song. This rule aims not to ensure the repetition of motifs, but to incentivize the composition of musical chunks which have the general format of a motif: a sequence of eight notes/events with three or more unique pitches. This is conceptually a good idea, but this formulation is evidently flawed since it does not reward the composition of motifs which may span more than eight notes in the composition due to the length of the notes or the existence of pauses. In any case, this reward function works to some effect and therefore we consider it to be appropriate even for meters different than the default 4/4.

9. **Play repeated motifs** - As mentioned in the description of the previous rule, the repetition of a given motif is frequently more important than its content. This is enforced through a reward function which simply checks whether there was a motif in the previous eight notes (including the action) and if so, check if it was present in its exact form throughout the whole previous composition. If so, the agent is rewarded with a substantial value. This implementation has severe limitations, namely the ones mentioned in the previous rule, in addition to the fact that it only detects the repetition of the exact same notes rather than also rewarding approximate repetition of the motif. Even still, it is quite ambitious and it attempts to solve one of the major issues in artificial music composition, as highlighted in 2.1.2, and therefore deserves some attention. In any case, it does not end up working very convincingly, since even in the demo published by Magenta [55], the percentage of notes in a repeated motif for the post-RL compositions was 0.0%. In our experience [5], this rule also did not manage to influence the outputs consistently.

In the end, the conclusions drawn from this analysis can be summarized briefly. This rule-set is useful since it partially eliminates common issues with RNN-based musical compositions such as excessive repetition, strangle leaps in pitch and, of course, notes not consistent with the apparent key of the piece. However, it suffers from some severe limitations such as being coded essentially only for the key of C Major and oversimplifying some of the reward functions, such as the ones related to motifs. *The most important issue, however, is that this rule-set does not explore more*

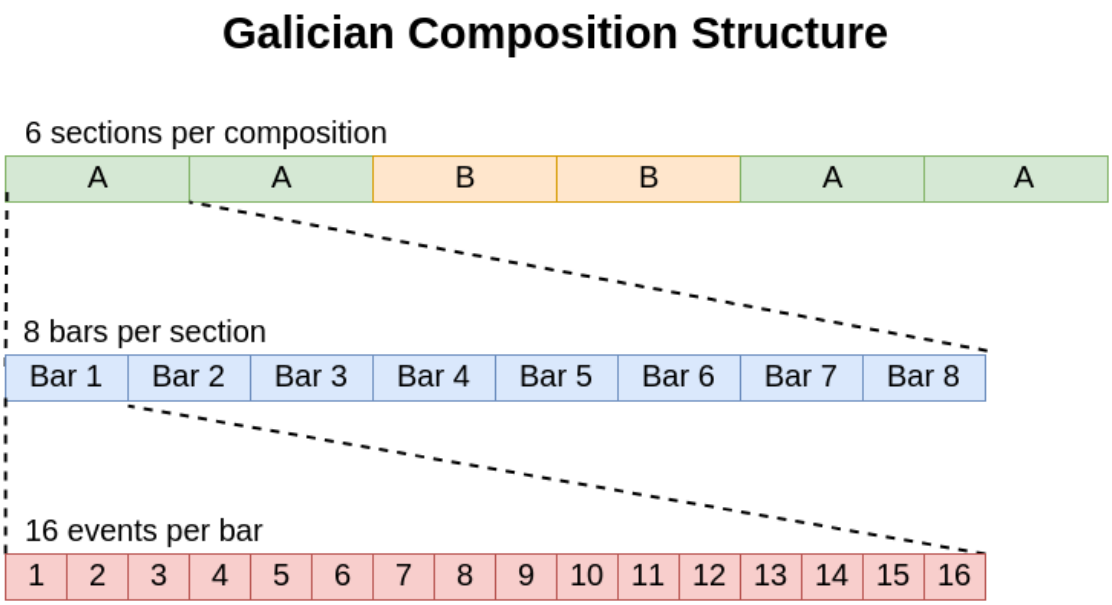


Figure 5.2: This scheme illustrates the format which we will be using for the RL Tuner compositions, based on the typical format of the pieces contained in the Galician dataset.

specialized rules which could steer the composition towards a more well defined style, which is what we aim to do in this project.

5.2.2 The Galician rule-set

This rule-set aims not only to tune the RNN, but to substantially influence its compositional style in order to (ideally) make even an RNN which was not trained with the Galician data-set compose in this style. In this way, what we mean to evaluate is the power of the RL Tuner and if it can be used for more than slight adjustments to the behavior of an RNN. The rule-set was elaborated by a musicologist (Emilia Parada-Cabaleiro) and a music psychologist (Eduardo Coutinho) who analyzed the dataset and its musical tendencies. In the end, it was decided that the compositional unit (the length of each element in the composition vector) should be the sixteenth note, since the RL Tuner automatically encoded the MIDIs from the dataset using this standard. A meter of 6/8 (12 sixteenth notes per measure) was adopted and the composition was divided into two sections, A and B, featuring different rule-sets. Each section contains 8 measures and each piece contains 6 sections, forming the sequence AABBAA. In terms of the sequential encoding, each section is made up of 96 notes/events, whereas the full piece contains 6 sections totaling 576 events.

After some preliminary experiments, it was decided that composing the full piece was overambitious for the RL Tuner, since it made the state-space and the variety of rewards so large that the learning process was very convoluted and time consuming. Therefore, the composition (and training) was performed section by section, using one of the two separate rule-sets each time. It is important to mention that the ap-

propriate keys for this genre are D Major, C Major, F Major, C minor and D minor, and the one we are following is defined as a parameter of the training procedure. It is also worth noting that, in the minor scales, one extra note is allowed (one semitone below the tonic) due to a rule which will be explained later. The general rules are the following:

1. **Sections should end with a quarter note** - Conceptually, this rule is quite simple: the composer should be rewarded for ending the composition with a quarter note. In practice, the reward is given when, in the last possible time-step of the section, the action (next note) is a held note event, the previous two notes are also held notes events and the note before these is either a note off event or a note.

2. **Sections should end with the tonic** - This rule's implementation has an interesting subtlety. The composition should be rewarded if the last note to be played is the tonic. A naïve approach to this reward function would be to reward the agent if and only if it selects the tonic note (for example, 14) at the last time-step. However, this is not a complete implementation of our requirement. The composition could be [...,14,1,0,0,1,0], where the last note would indeed be the tonic but no reward would be attributed. Therefore, the reward function must take these cases into account by, at the last time-step, checking whether the action corresponds to the tonic note *or* if the action is a held note/pause and the latest note in the previous composition was the tonic. This kind of thought process is extremely important in order to implement these rules properly. The issue here is that by providing the reward in the end for a tonic which was played five notes before the end of the composition, we distance the time-step in which the reward is attributed from the time-step in which the tonic was played, which can complicate the learning process. However, in this case, as well as in many others, this is effectively the only way of programming the reward accurately.

3. **The sub-tonic should be followed by the tonic** - In other words, the note which is one semitone below the tonic note (the sub-tonic) should be followed by the tonic itself. This is relatively simple to implement: if the most recent note was the sub-tonic, selecting the action corresponding to the tonic should entail a positive reward. In this case, the issue is that we risk biasing the composer towards constantly composing sub-tonics simply to then compose the tonic and get the reward, which is undesirable. Because of this, the value of the reward should be fine tuned to make it clear that it is effective without saturating the piece.

4. **After a leap, play the immediate opposite note** - This rule was noted after observing the following behavior in the dataset: after a substantial leap in pitch in a certain direction (a fourth or larger), the next note is usually the note directly below or above the current note, opposing the direction of the leap. Specifically, this was then implemented as a reward function by detecting leaps of more than 3 notes in the scale (a fourth or larger interval, checking if both

notes are in the scale) and then rewarding the agent for choosing the appropriate action (one which is one semitone or less in the opposite direction and is in the scale). The problem of saturating the composition with this behavior is a concern, just as it was for the previous rule.

5. **Reverse direction after multiple notes in the same direction** - By listening to some of the pieces in the dataset, one of the aspects that becomes apparent is that they typically feature many ascending and descending melodies which are relatively short and generally stabilize by returning to the original tonal range. We found that the best way to implement this was to reward the agent for choosing a note in the opposite direction after having composed a sequence of notes all following the same direction (three or more notes, where the reward is proportional to the length of the sequence). This stops the network from creating sequences that are too long without varying the pitch contour (the pitch variation throughout the song), while also implicitly incentivizing the agent to compose these sequences which, as mentioned previously, should be prevalent in this genre. The issue of saturating the composition must also be kept in mind, but it is generally less worrying than in previous cases since this reward is less oppressive.

The section-specific rules are described as follows:
Section A:

1. **Note lengths** - Section A should contain faster notes, which means that it should feature sixteenth notes and eighth notes in approximately equal parts, and only one or two quarter notes. It should not contain any longer notes. The rewards for note lengths are attributed in the following way: if, during the time-step after our note, the previous composition consists of the note we want to reward ([16,0], for instance, if we want to reward eighth notes), then the agent should be rewarded for choosing any action other than the held note event, ensuring that the composition will indeed contain an eighth note and not a longer note. This method is slightly counterintuitive but it is the most direct way of rewarding this behavior.

2. **Intervals** - Thirds should be the most common interval in section A. This is ensured by using a reward function that allocates a positive reward when the agent composes a note which forms a third when paired with the previous note, and both of the notes are in the scale which is being followed. The risk of degenerating into an agent which only composes using this interval is quite high, and therefore the value for this reward must be taken into consideration.

3. **Start with a quarter rest** - This rule is implemented by attributing a positive reward when the composition is made of 4 held note events after the first 4 time-steps. This signifies a quarter rest, given that no other note has been played. The implementation is similar to rule A1.

Section B:

1. **Note lengths** - Section B should contain slower notes, which include quarter notes and eighth notes. Sixteenth notes should be avoided, and therefore warrant a negative reward. The implementation is similar to rule A1.

2. **Intervals** - Large intervals should be prevalent. Therefore, a positive reward is emitted whenever an interval larger or equal to a fourth is composed. The implementation is similar to rule A2.

3. **End with a quarter note and two quarter rests** - The idea here is that the last bar of section B should consist of a quarter note in the beginning and then be silent. The last silence can be seen as a half note rest, two quarter note rests, four eighth note rests, etc. Specifically, the first quarter note is rewarded in the form we have already described, and the silence is rewarded in two ways: if the agent composes a quarter rest in the correct time-step, it receives a very heavy reward; and if it selects the note off event as an action throughout the last two thirds of the bar, it receives a relatively small positive reward. This final silence can be rewarded in many different ways, but it is challenging to achieve the desired behavior, as we will mention later on.

A note about the rules which were discarded from the final rule-set:

Two of the rules which we initially formulated were not kept for the rule-set used in the full experiments:

1. **Intra-section repetition** - This is probably the most conceptually interesting rule that has been presented so far, but it is plagued by some very large issues. Essentially, the intended behavior is that each section's pitch contour should be symmetrical, in the sense that the first four bars should have a similar contour to the last four. This does not necessarily mean that their rhythm must be similar, but only that the relative "ups and downs" of the first half should be recognizably similar to the second half. The first issue is that this behavior is empirically quite difficult to implement in a form which can be easily learned. We could, of course, at the end of each section, calculate the similarity of the contours and reward the network accordingly. However, this alienates the reward from the actions that led to it, which is very detrimental since it makes it very difficult for the Deep Q Learning architecture to learn how to maximize this reward. We can, instead, opt to check the intervals of the second part compared to the first, action by action, and apply the reward step by step in order to make the reward process and therefore the learning smoother. However, the larger issue here is that this rule naturally incentivizes repetition and correlation between the current notes and the previous composition. This is a very dangerous behavior since it can lead to the unnatural repetition which we tried to deemphasize by using rules 3 and 7 from the original rule-set. Mainly due to this reason, this rule/reward was not included in the final rule-set.

2. **Note repetition is allowed with quarter notes, especially in section B** - The way to implement this would be to decrease the penalty for repetition if it is

done using quarter notes, and decrease it further if it is also in section B. As with the previous rule, this would allow the network to be more repetitive, which is clearly not wise. Even though this behavior was observed in the original dataset, we decided it was best not to associate it with a reward due to its potential for disturbing the final compositions.

#	Rule name	Original Rule-set	Galician rule-set	Combination of rule-sets
1	Stay in key	✓		✓
2	Start/end with tonic	✓		
3	Avoid repetition	✓		✓
4	Prefer harmonious intervals	✓		✓
5	Resolve large leaps	✓		✓
6	Avoid repeating extrema	✓		
7	Avoid high autocorrelation	✓		✓
8	Play motifs	✓		✓
9	Play repeated motifs	✓		✓
10	End with a quarter note		✓	✓
11	End with the tonic		✓	✓
12	Follow sub-tonics with tonic		✓	✓
13	After a leap, play the opposite note		✓	✓
14	Reverse direction		✓	✓
A1	Compose fast notes		✓(A)	✓(A)
A2	Compose thirds		✓(A)	✓(A)
A3	Start with quarter rest		✓(A)	✓(A)
B1	Compose slow notes		✓(B)	✓(B)
B2	Compose broad intervals		✓(B)	✓(B)
B3	End with quarter note/two quarter rests		✓(B)	✓(B)

Table 5.1: This table explains the three different rule-sets explored in this chapter, showing which rules are contained in which rule-set. Checkmark (A) and checkmark (B) represent the rules that are specific for each section (A or B).

5.2.3 Combining the two rule-sets

In order to attempt better results, we decided to combine the original and the new dataset into a mix of the two. This combined rule-set contains every rule in the new rule-set and most of the rules from the original set. The exceptions were rule 2 (start and end with the tonic note), since we were not satisfied with the restrictive implementation of this rule and the idea behind it did not match our new genre; and rule 6, since unique high/low notes were not associated with this genre. In addition, some modifications were made to the original rule-set in order to properly suit new keys other than C Major. In order to test the adequacy of these changes, we ran two experiments in which only the notes in the key of C# Major was rewarded (which only has two notes in common with C major) using this rule-set. These featured between 1 and 3 % notes off key [5], which generally means that the code now works well for any major key.

In short, the purpose of this rule-set is the same as the Galician one: steer the composition towards the *Xota Gallega* style. However, it also includes some general rules which can help improve outputs, namely the ones which aim to alleviate the issue of repetition. Theoretically, this rule-set should yield the best results.

5.3 Further details regarding training procedure and tuning

Training the RL Tuner itself also involved tuning a fair amount of hyperparameters. Here are the ones that were most relevant:

- **Note RNN type** - As we have stated, the only type of RNN we will be using in this project is the Basic RNN from Google Magenta, with some additional modifications for the New Attention RNN.
- **Number of training steps** - This value was originally set to 1000000 steps for the RL Tuner. After much experimentation with lower values in order to speed up the training process, we decided that using a significantly lower amount of steps made the training far too inconsistent, and therefore was not worth it. To be clear, we left this paramter at its original value for the remainder of the project.
- **Algorithm** - This parameter refers to the three different possible RL algorithms which this architecture supports: Q-learning, Psi-learning and G-learning. As mentioned in 2.1.1, this will be set to Q-learning throughout the whole project since this is by far the most tried and tested of the three approaches.
- **Exploration mode** - The RL Tuner architecture is compatible with Boltzmann exploration (which samples from the model's output to select the next action) and ϵ-greedy exploration (mentioned in section 2.1.4). Since the latter had seen far more use in the context of RL and yielded good results, we decided to adopt it.

- **Random action probability** - In our case, since we are using ϵ-greedy exploration, this value is commonly known as ϵ. Its default value was 0.1 and, due to the annealing mentioned below to encourage initial exploration, this somewhat low value seemed reasonable and was kept.

- **Exploration steps** - This value represents the number of training steps (starting from the beginning of training) which will be taken until the exploration rate (ϵ) reaches its final value (0.1). In order to encourage exploration during the initial training steps, ϵ is set to 1 on the first training step. This means that, at this point, the exploration is absolutely random. As the training advances, this value is annealed towards the value we set for ϵ as a hyperparameter. By setting the number of exploration steps to half of the training steps (500000), we are configuring the network to perform this annealing for the first half of this training and then proceed through the second half with the same value for ϵ until the end. The default value for this hyperparameter was indeed 500000 and was not changed.

- **Reward mode** - This parameter controls the rule-set that will be used for the training and can be set to five distinct configurations (three of which did not exist in the original project):

  - no_rl - there is no rule-set for training, since we are only going to generate without training.
  - zero_reward - The reward for music theory is set to zero at all times.
  - music_theory_only - Uses the original rule-set (5.2.1).
  - galician_only - Uses the Galician rule-set (5.2.2).
  - both_rule_sets - Uses the mixture of the original and the Galician rule-sets (5.2.3).

- **Reward scaler** - This determines the weight which is placed on the music theory reward, i.e.

$$\text{Total\_reward} = \text{Reward\_scaler} * \text{Music\_theory\_reward} + \text{RNN\_reward} \quad (5.1)$$

  This was set to 1.0 originally, but due to the high emphasis of the music theory reward in our research, we decided to set it to 2.0 instead.

- **Attention** - Should be set to "True" if we are loading a checkpoint from the New Attention RNN. This was not in the original project.

- **Mini-batch size** - This refers to the amount of samples from the experience buffer used at each training step (this term was introduced in 2.1.1). The default value for this was 32 and since this led to good results and we did not have any memory issues during training, it was kept this way.

- **Discount rate** - This parameter was mentioned in 2.1.4 and it controls how much the agent values future rewards. It is set to 0.5 and was kept at this value.

- **Target network update rate** - As mentioned in 2.1.4, the Target Q Network should be updated frequently to be similar to the Q Network. This is done by applying this update function to all of the variables in the Q Network/Target Q Network:

$$target = (1 - \alpha) * target + \alpha * source \tag{5.2}$$

  where target is the variable from the Target Q Network, source is the variable from the Q Network and α is the update rate. The value for this parameter was kept at 0.01, as in the original code.

- **Key** - The key in which the agent should be trained to compose. This is set to C Major as default, and it was kept this way for almost the whole project, apart from some specific experiments.

- **Number of notes in melody** - As mentioned in 5.2.2, we compose section by section so the number of notes is 96, as opposed to the original 32.

Apart from the training configuration, there is one aspect in this procedure which is worth exploring in depth. One of the biggest challenges in this project was the inconsistency of Deep Q Learning applied to music composition. Deep Learning methods often have trouble with inconsistency due to the generalized use of stochastic processes in order to update the network weights efficiently, as highlighted in 2.1.1. This unreliability is exacerbated by the exploration algorithms typically used in RL, namely ϵ-learning, which makes the final policy extremely dependent on the initial random exploration. For these reasons, and due to the fact that we are using a trained RNN for part of the reward, which is already known to be quite unpredictable in its compositions, the outputs post-RL training can have a high intra-procedural (compositions coming from the same trained network) as well as inter-procedural (compositions from different networks trained with the same parameters) variance.

The intra-procedural variance is to be expected in this domain (since any musical dataset can feature a high variance), and this can be counteracted by composing a large amount of compositions in order to get representative statistics. The inter-procedural variance, however, is heavily accentuated, especially with heavy music theory rewards, and cannot be completely fixed by increasing the number of training steps. Table 5.2 portrays this issue with concrete training results, by comparing the composition statistics between two models trained with the exact same parameters. The only way of mitigating this problem to a certain extent is to perform various training procedures with the same exact configuration. However, this is not an ideal solution since it evidently increases the amount of time needed to get the final results. In any case, this methodology will be applied to one configuration in chapter 6.

	1st run	2nd run
avg. num. sevenths per composition	0.026	0.151
avg. num. fifths per composition	3.077	1.078
avg. num. rest intervals per composition	**12.37**	**0.147**
avg. num. of eighth notes in composition	**18.01**	**8.893**
avg. num. of late quarter notes in composition	0.191	0.374
avg. num. of late pauses in composition	**0.23**	**0.005**
avg. num. of last quarter notes in composition	0.171	0.371
avg. num. of sub tonics in composition	0.017	0.007
avg. num. of opposite seconds in composition	0.446	1.036
avg. num. of direction changes in composition	**0.973**	**3.328**

Table 5.2: This table mentions some of the compositional statistics of two trained RL Tuner models, trained with the exact same parameters, RNN checkpoints (pre-trained Basic RNN) and rule-sets (original+Galician). Although they were trained for 1000000 steps, we can still see that there are some substantial differences in statistics which should be tuned by specific rules. This is a clear example of how unreliable the results from a single training run can be with this architecture. The statistics were calculated by generating 1000 compositions totaling 96000 notes. It is worth noting that not all training runs feature this large amount of variance, since this is an extreme case.

5.4 Preparing and modifying the original RL Tuner (Implementation details)

The code for the original RL Tuner was made in 2016, receiving some additional updates throughout the years. However, after we acquired the open-source code and ran it with the most recent version of Tensorflow (which at the time was version 1.8.0), it was apparent that the model was not functional. This was due to two main concerns. The first of these was the fact that the names of the variables in the pre-trained checkpoints differed slightly from their names in the actual Tensorflow graph (this was the case for both the note RNN and the Basic RNN). It is likely that the pre-trained checkpoints were simply outdated when compared to the variable names automatically created by Tensorflow when the RNN cell is created. The solution to this issue involved analyzing the checkpoints and writing scripts to change the variable names to the appropriate strings, adapting code originally obtained from [55]. The second issue was substantially more complex. The LSTM state is a integral part of the RL Tuner architecture since it is used as the state of the MDP. The issue lies in the fact that the format of the state in Tensorflow has changed since 2016. Specifically, it used to be a simple vector of vectors, containing all the information regarding the current state, while nowadays it has been updated to be a vector of LSTM State Tuples, which are Tensorflow objects made specifically for this purpose. This change meant that the original code was made obsolete and did not compile. Therefore, some technical work had to be done to adapt the code to this new state format. This involved a fair amount of modifications to some core elements of the original work, which can be found in [5].

After these changes, the project was compiling properly. However, some of the statistics gotten after performing the training with the same parameters as in the Magenta notebook demo [55] were slightly different, evidently due to the fact that the demo had used the Note RNN, whereas we were experimenting with the Basic RNN. This meant that we had to change some rewards slightly, namely since the compositions were lacking pauses, but the rewards and their values stayed largely the same and produced results comparable to those presented in [2]. After the results of the paper were approximately replicated, the code had to be modified to accommodate Attention Mechanisms. This involved changing the way the LSTM state was parsed and fed to the Q Network, since the state gains a new dimension in the New Attention RNN. This is due to the fact that an Attention wrapper is added to the LSTM cell. This process also included some checkpoint manipulation, but this was mentioned in the previous section. Apart from this, some minor quality of life changes were applied to the original code and the file which is used to train the RL Tuner was heavily modified, gaining new flags and functionalities in order to suit the interests of this project. An example of one of the new features is recording statistics pre and post training in order to directly evaluate the impact of the RL training in the compositions. A flag was also added to inform the network whether we are dealing with attention mechanisms or not, as this will require a slightly different procedure.

Chapter 6

Experiments and analysis of results

In this chapter we outline the experiments performed in this project and analyze their results. We begin by summarizing which experiments were performed, defining the RNN/rule-set configurations that were tested. We then analyze the results generated by the RNNs before any RL training, observing the statistics that were measured by the model (which can be seen in detail in the project repository [5]) as well as the behavior in the MIDI samples (found in [5]) to determine the quality of the compositions, and use them as a baseline for the next experiments. After this, we analyze our main configuration, the Pre-trained Basic RNN trained with Both Rule-sets (original and Galician). In this section, we delve into detail regarding the post-RL compositional statistics and how they compare with the pre-RL statistics, in order to measure the effectiveness of every reward in this rule-set. In the next two sections, we discuss the behavior of other RNNs trained with both rule-sets, as well as every RNN with the remaining reward modes (zero reward, original and Galician). We finish by using a new graphical representation to compare the pre-RL and post-RL statistics of every tested configuration and commenting on how this representation illustrates some of the statements that were made during the analysis of the experiments.

6.1 Outline of the experimental procedure

For the final training procedures, we decided to test every RNN configuration mentioned in 4.5 (Pre-trained Basic RNN, Galician Basic RNN, Pre-trained+Galician Basic RNN,Galician Attention RNN) with all 5 reward modes mentioned in 5.3., in order to present as many relevant results as possible. The pre-trained Basic RNN tuned with both rule-sets was run five times, since it is the main object of our analysis, whereas every other variant was trained only once and will be analyzed more briefly. *The reason why this is our main configuration is the fact that the new rule-set was tuned towards this combination of RNN and reward mode.* Specifically, the pre-trained RNN was preferred due to its malleability (reacting well to RL tuning since it is not overconfident about the next note) given the broad dataset with which it was trained, and the original+Galician rule-set was used due to the fact that it appeared to be the most well-rounded rule-set available (it prevents common problems

in RNN composers while still steering the composition towards a particular style). The key used for the rule-set was C Major and the augmented version of the dataset was used for all relevant configurations. **The results of the experiments, including the statistics and the generated samples, can be found in the project repository [5]**.Below we describe every configuration that was tested:

	No RL	Zero Reward	Original Rule-set	Galician Rule-set	Both Rule-sets
Pre-trained Basic RNN	1.1	1.2	1.3	1.4	1.5
Pre-trained+Galician Basic RNN	2.1	2.2	2.3	2.4	2.5
Galician Basic RNN	3.1	3.2	3.3	3.4	3.5
Galician Attention RNN	4.1	4.2	4.3	4.4	4.5

Table 6.1: This table denominates all the experiments performed in this chapter for future reference

6.2 Melody RNN, No RL Training

In this section we discuss the outputs generated by the RNNs before the Reinforcement Learning, in order to display a clear picture of their compositional style, which can then be compared to the other configurations.

6.2.1 Pre-trained Basic RNN (1.1)

We begin with the Pre-trained RNN. This network has two main issues. The first of these is excessive repetition, which is a common issue for RNNs but is even more prevalent than usual for this network, featuring 22.46 % excessively repeated notes. This can easily be noticed by listening to the samples [5], which often repeat the same note for the whole piece. The second issue is the lack of adherence to a key. Many compositions have no clear key guide, which makes it abundantly clear that they were not composed by a trained musician. This is mostly due to the fact that the dataset used for training is extremely broad, but it is still unacceptable. Apart from these problems, the compositions feature decent compositional statistics but sometimes have notes which are clearly too long or very long periods of silence which sound unnatural.

6.2.2 Galician Basic RNN (2.1)

The next RNNs were trained internally, using checkpoints resulting from our training process and not taken from Magenta [55]. Due to the overfitting issue detailed in section 4.4, the accuracy of the models was monitored and three checkpoints were produced for each of them, featuring 85 %, 90 % and 95 %. The idea here was to

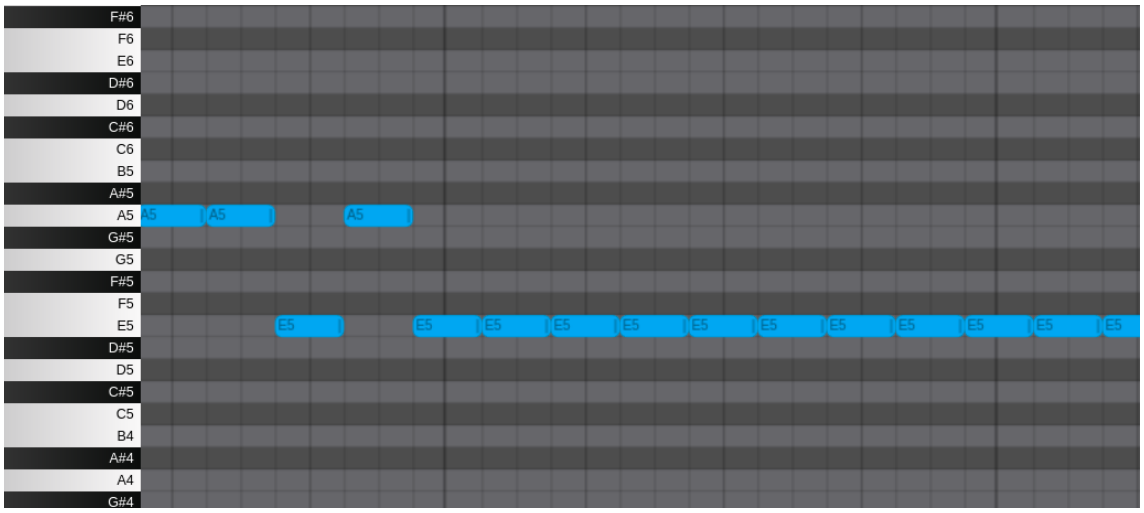


Figure 6.1: This image is taken from the piano-roll representation of a sample generated by the Pre-trained Basic RNN (1.1). This situation highlights the issue of excessive repetition.

experiment with lower accuracies to prevent overfitting and make the reward RNN have a more diverse rewarding function. In the end, it is noticeable in the note probability density charts (Figure 4.2) that the network is less overconfident about the next note, which is exactly what is required in this case. The first RNN trained with the Galician dataset was the Basic RNN (with no previous training). The compositions still suffer from excessive repetition, but to a much lesser extent (between 2 and 5 %). The repetition happens relatively rarely but once it does the network clearly gets into a loop where it chooses the same note as the next note multiple times, and it noticeably harms the quality of the piece. In every other regard, the compositions are acceptable. They generally stay in the same key and tonal range, and they feature the ascending/descending pitch contour which appears frequently in the Galician dataset. The differences between the three different checkpoints in terms of compositional style are minor and the previous comments apply to all of the accuracies mentioned above.

6.2.3 Pre-trained+Galician Basic RNN (3.1)

After this, we also attempted using the pre-trained RNN and training it with the Galician dataset on top of the original training. This worked relatively well, but the repetition tendency of the pre-trained network persists, reaching almost 10 % excessively repeated notes. Although this model achieves high training accuracy in few training steps, its compositions do not appear to embody the Galician dataset due to an accentuated frequency of very long notes, among other issues. The RNN with 95 % accuracy seemed to produce the best outputs, although they still suffer from the same issues.

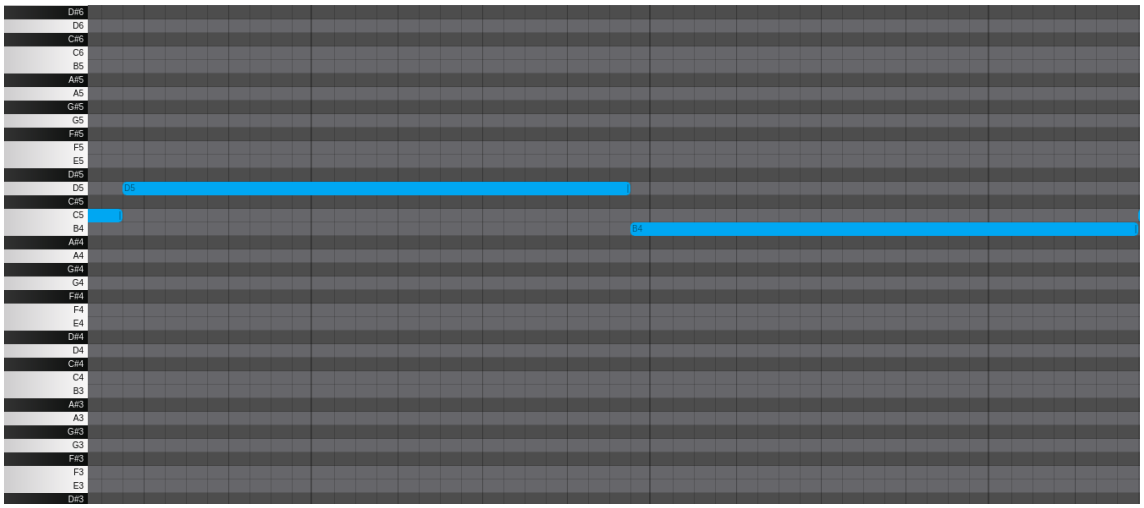


Figure 6.2: This image is taken from the piano-roll representation of a sample generated by the Pre-trained+Galician Basic RNN (3.1). In this case, we can clearly see that the network has composed two notes which are too long (longer than a whole note).

6.2.4 Galician Attention RNN (4.1)

The final RNN configuration was the New Attention RNN, trained only with the Galician dataset. This network takes far longer (in terms of training steps) to reach an acceptable accuracy, but it does not yield noticeably better results than the Basic RNN trained with the same dataset. The compositions are slightly more inconsistent than the ones produced by the Basic RNN, containing overly long notes and pauses at times. However, it must be said that the best pieces produced here are more impressive than what is seen in any of the pre-RL previous configurations, although not by a large margin. In the end, the addition of attention mechanisms is not an evident upgrade over the original network, but it does lead to compelling results in some cases.

6.3 Pre-trained Basic RNN with Both Rule-sets (1.5)

In this section we compare the compositional statistics pre and post-RL training for the pre-trained Basic RNN with the original+Galician rule-set, which is our main configuration as mentioned in 6.1. We separate this analysis in sections since there is a separate rule-set/reward mode for each of them.

A note about the following sections:

The results which will be mentioned below refer to three statistics:

- Pre-RL - This refers to the statistic measured in 1000 compositions generated by the RNN before any RL training. The mean and standard deviation are presented to summarize the five values obtained after the five separate training procedures.

- Post-RL - This refers to the statistic measured in 1000 compositions generated by the RNN after the RL training (with both rule-sets). The mean and standard deviation have the same meaning as the Pre-RL statistic.
- Best value - This refers to the best value out of the five Post-RL values. If we are in section B, for instance, and we are measuring the average amount of sixteenth notes per composition, the best value would be the lowest of the five values.

It is worth noting that the rule numbers used in this section are different from the ones used in chapter 5, since we are specifically speaking about the Original+Galician rule-set and the rules are organized by order of importance for the analysis.

6.3.1 Section A

I **Stay in key** - This rule worked very well, which was expected since it was already a success in the original paper, but given that we are using a different RNN and adding the new rule-set this is quite a satisfying result. This reward is effective since notes in and out of key happen often in the exploration, leading to a smooth training procedure. Also, the reward is applied directly to the action, which leaves no distance between the action that caused the reward and the application of the reward, which is desirable in any Reinforcement Learning environment.

Pre-RL	Post-RL	Best value
$2.71 \pm 0.17\%$	$\mathbf{0.85 \pm 0.29\%}$	**0.32%**

Table 6.2: Percentage of notes not in key (not contained in the set which represents the key of C Major)

II **Avoid excessively repeated notes** - This rule also worked remarkably well, as expected. This is perhaps the most crucial of the original rules and it is extremely important that excessive repetition is almost non-existent post-RL.

Pre-RL	Post-RL mean	Post-RL
19.71 ± 0.68 %	$\mathbf{0.001 \pm 0.002}$ **%**	**0.0%**

Table 6.3: Percentage of notes repeated excessively

III **Note lengths** - Getting the correct note lengths was one of the main challenges of this training process, since this is a fundamental part of any musical composition and it was not attempted in any way in the original paper [2]. In this section the note lengths should be focused on the eighth notes, while having also a high amount of sixteenth notes. There should also be a somewhat reduced amount of quarter notes. We can see by the post-RL statistics that this objective was

achieved with no issues, removing all of the notes longer than a quarter in length since these are not prevalent in the galician dataset. One could argue that the distribution of note lengths is similar between the original RNN compositions and the final compositions, but *the increase in eighth notes and the elimination of any longer notes clearly show that the rewards of the RL Tuner have complete control over this aspect of the compositions, even if it requires a fair amount of manual tuning. This is a remarkable achievement for the project.* The control is further shown through the fact that the frequency of sixteenth notes is roughly the same pre and post-RL, rather than increasing exponentially as was the case for the original rule-set (featuring almost 50 sixteenth notes per composition on average).

Note length	Pre-RL	Post-RL	Best value
1/16	17.63 ± 0.66	$\mathbf{15.26 \pm 3.01}$	**16.95**
1/8	9.78 ± 0.40	$\mathbf{34.23 \pm 3.60}$	**39.45**
1/4	2.76 ± 0.14	$\mathbf{2.13 \pm 1.39}$	**2.79**
1/2	0.79 ± 0.07	0.0 ± 0.0	0.0
1	0.29 ± 0.03	0.0 ± 0.0	0.0
>1	0.23 ± 0.03	0.0 ± 0.0	0.0

Table 6.4: Statistics regarding number of notes with specific note lengths per composition

IV **Intervals** - [1] In this section, the objective was to have an increased amount of thirds and keep the other intervals at a reasonable level in order to avoid saturating the compositions. We can see that *the number of thirds increased by more than four times after the RL training*. This clearly indicates how successful this rule was. However, if we look at the percentage of thirds, we can see that, on average, thirds constitute less than half of the total intervals. This shows that the intervals in most compositions are still varied, which is most likely due to the addition of the original rule-set.

V **Resolve large leaps** - For this rule, we can see that most leaps are already resolved in the original RNN (around 75%). This percentage does not rise substantially after applying the reward, which is acceptable due to its already high value. However, the number of leaps definitely increases, which is a side-effect of the reward and not its initial intention. This interaction shows how some rules can influence statistics in a way which is not necessarily expected, but can still be desirable. In any case, the increased amount of leaps may also be related to the rule added in the galician dataset which also incentivizes this kind of behavior.

VI **Avoid high auto-correlation** - For this statistic, we take the absolute value of the correlation (which can be positive or negative) in order to obtain relevant

[1] To produce the results shown in the table regarding intervals, we performed a new training run with new statistics (separate from the run which produced the rest of the results displayed in this chapter), but kept the rest of the code unaltered (including the parameters and reward functions). This was due to the fact that the statistics regarding intervals in the original project were incomplete.

Interval	Pre-RL	Post-RL	Best value
non-preferred major thirds	1.33 ± 0.03	2.03 ± 0.95	3.77
preferred major thirds	0.75 ± 0.03	**9.61 ± 1.53**	12.29
minor thirds	3.08 ± 0.09	**10.26 ± 1.25**	12.39
total thirds	5.16 ± 0.13	**21.89 ± 1.14**	22.81
percentage of thirds	20.21 ± 0.31 %	**48.86 ± 2.66** %	53.00 %

Table 6.5: Statistics regarding number of intervals per composition

Name of statistic	Pre-RL	Post-RL	Best value
Percentage of leaps resolved	75.18 ± 0.96 %	75.76 ± 3.80 %	80.39 %
Number of leaps resolved	0.73 ± 0.02	**2.45 ± 0.33**	**2.94**

Table 6.6: Statistics regarding leaps larger than a fifth in compositions

values for the mean and standard deviation. By looking at the data below, it is apparent that the correlation did not reliably decrease with training, having even increased in two of the three measures. This is not desirable, but the fact that these values did not spike is what is important. The correlation can be high, for instance, due to the existence of a lot of held notes (the correlations when the composition is [0,0,0,0,0,0,0,0] are all very high since the previous compositions with lags 1, 2 and 3 are equivalent to the current composition). Due to this reason and also the high variance featured below, it is important to have a critical mindset when analyzing this statistic and listen to the composed samples in order to understand if this correlation is detrimental (which is not the case).

Name of statistic	Pre-RL	Post-RL	Post-RL
Correlation with lag 1	0.18 ± 0.01	0.33 ± 0.10	0.17
Correlation with lag 2	0.17 ± 0.02	0.23 ± 0.13	**0.003**
Correlation with lag 3	0.16 ± 0.01	0.12 ± 0.10	**0.01**

Table 6.7: Statistics regarding correlation in compositions

VII **Play (repeated) motifs** - As mentioned in the Background, motifs are fundamental for proper music composition. It is therefore encouraging to see that a large percentage of notes constitute motifs after the RL training. However, it is important to take into account that a motif is represented here as a group of 8 events of which 3 are unique, which is evidently not uncommon. Also, repetition of motifs is extremely rare even after training, probably due to the fact that this is a very difficult behavior for the network to learn, regardless of the reward value associated with it. In any case, rewarding this type of patterns

is conducive towards more compositional variety, which is very important for the musicality of the pieces. Additionally, the difference between pre and post training compositions is substantial, which is excellent.

Name of statistic	Pre-RL	Post-RL	Best value
Notes in motif	22.95 ± 0.65 %	**77.31 ± 2.14 %**	**79.72 %**
Notes in repeated motif	0.22 ± 0.05 %	0.88 ± 0.61 %	1.95 %

Table 6.8: Statistics regarding motifs in compositions

VIII **End with a quarter note** - Although this rule is not able to affect a very large number of compositions, the post-RL composer is more than twice as likely to compose a piece ending with a quarter note than the pre-RL composer. This means that our reward methodology applied here is working fairly well, especially given that the number of quarter notes is roughly the same before and after training. This is important because if the number of quarter notes had increased, the network could have been introducing a bias and composing more quarter notes randomly in order to increase the chance of getting rewarded in this way, instead of learning the intended behavior. Since this is not the case, we can confirm that the reward is leading to the intended result.

Pre-RL	Post-RL	Best value
0.77 ± 0.27 %	**1.60 ± 1.36** %	**3.80** %

Table 6.9: Statistics regarding the average percentage of compositions which end with a quarter note

IX **End with the tonic** - This rule was clearly successful, affecting roughly one third of compositions after one of the runs. This is another remarkable result since the behavior it rewards is not linear, i.e., the reward is not attributed at an obvious or consistent time-step.

Pre-RL	Post-RL	Best value
15.91 ± 1.32 %	**29.62 ± 2.21** %	**33.00 %**

Table 6.10: Statistics regarding the average percentage of compositions which end with the tonic note

X **The sub-tonic should be followed by the tonic** - We can see that this rule also led to the desired outcome. In this case the reward is applied directly, i.e., at a certain state (when a sub-tonic has been played) the network gets an immediate reward for playing the tonic note. This is the main reason behind its clear success.

Pre-RL	Post-RL	Best value
0.53 ± 0.04	$\mathbf{3.94 \pm 1.08}$	**4.95**

Table 6.11: Statistics regarding the average number of sub-tonics followed by tonics per composition.

Pre-RL	Post-RL	Best value
0.69 ± 0.02	$\mathbf{2.90 \pm 0.45}$	**3.38**

Table 6.12: Statistics regarding the average number of opposite seconds after leaps (intervals greater or equal than a fifth) per composition.

XI **After a leap, play the opposite note** - This is another impactful rule, which works for the same reasons as the previous one.

XII **Direction changes** - This rule is implemented in a complex fashion and is generally quite difficult to describe, let alone learn. Regardless, due to the fact that the application of the reward is simultaneous with the principal behavior that it means to promote (the direction change), the function is able to affect the final compositions in a meaningful way.

Pre-RL	Post-RL	Best value
2.67 ± 0.13	$\mathbf{9.06 \pm 1.64}$	**12.04**

Table 6.13: Statistics regarding the average number of changes in pitch direction after three or more of the last six intervals in the same direction per composition.

XIII **Begin with a quarter rest** - *This rule was without a doubt the most successful out of all the novel rewards tested*. This was due to its relative simplicity and consistency. However, the reason why this rule worked very well compared to, for instance, the rule regarding late quarter notes, sheds light onto the methodology that leads to impactful reward functions in the context of music composition which will be discussed in the conclusion.

Pre-RL	Post-RL	Best value
2.66 ± 0.44 %	$\mathbf{88.64 \pm 2.95}$ **%**	**93.20 %**

Table 6.14: Statistics regarding the average number of changes in pitch direction after three or more of the last six intervals in the same direction per composition.

6.3.2 Section B

Rules I,II,VII,IX,X,XI mentioned in 6.2.1 for section A worked similarly well for section B, as we can see in table X, and therefore do not warrant further analysis. Rule XIII does not apply to section A. We proceed to analyze the rules which lead to different results with the section B rule-set, and explain why these results make sense in this context:

Name of statistic	Pre-RL	Post-RL	Best value
Notes off key	2.71 ± 0.17%	**0.19 ± 0.15 %**	**0.03 %**
Notes excessively repeated	19.71 ± 0.68 %	**0.12 ± 0.15** %	0.00 %
Notes in motif	22.95 ± 0.65 %	36.82 ± 24.74 %	71.21 %
Notes in repeated motif	0.22 ± 0.05 %	2.24 ± 1.63 %	4.47 %
Compositions ending with tonic	15.91 ± 1.32 %	**38.16 ± 13.06** %	**56.0 %**
Number of sub-tonic/tonic sequences	0.53 ± 0.04	**1.15 ± 0.82**	**2.06**
Number of opposite seconds after leap	0.69 ± 0.02	**1.73 ± 0.82**	**2.91**

Table 6.15: Statistics regarding rules I,II,VII,IX,X and XI in compositions

III **Note lengths** - Section B should have slower notes, having a high number of quarter notes and octave notes. No other note length should be prevalent. This is clearly achieved, and the steep increase in quarter notes shows the power of the rewards since longer note lengths are more difficult to reward, as mentioned previously. The elimination of sixteenth notes is also extremely effective.

Note length	Pre-RL	Post-RL	Best value
1/16	17.63 ± 0.66	**0.02 ± 0.01**	**0.01**
1/8	9.78 ± 0.40	**20.56 ± 14.16**	**17.98**
1/4	2.76 ± 0.14	**13.69 ± 7.07**	**23.55**
1/2	0.79 ± 0.07	0.0 ± 0.0	0.0
1	0.29 ± 0.03	0.0 ± 0.0	0.0
>1	0.23 ± 0.03	0.0 ± 0.0	0.0

Table 6.16: Statistics regarding number of notes with specific note lengths per composition

IV **Intervals** - [2] In this section, intervals equal to a fourth or above should be prevalent. This is clearly achieved for every interval in this range, while still having a fair amount of seconds and thirds so that the composition does not become saturated with large intervals.

V **Resolve large leaps** - In this case, we see that the number of resolved leaps is similar to section A, while the percentage of resolve leaps is lower. This may be due to the fact that leaps are rewarded as intervals in this section or perhaps due to the fact that notes are generally more distant in this section, which makes specific pitch behaviors more difficult to learn.

[2] As in section A, we performed a new training run with new statistics to obtain these results.

Interval	**Pre-RL**	**Post-RL**	**Best value**
perfect fourths	2.54 ± 0.07	$\mathbf{6.38 \pm 2.78}$	11.62
diminished fifths	0.43 ± 0.03	0.73 ± 0.70	1.63
non-preferred perfect fifths	1.01 ± 0.06	1.36 ± 0.85	2.86
preferred perfect fifths	0.56 ± 0.05	$\mathbf{3.59 \pm 1.28}$	4.85
minor sixths	0.43 ± 0.04	$\mathbf{0.88 \pm 0.31}$	**1.45**
major sixths	0.46 ± 0.03	$\mathbf{2.69 \pm 1.27}$	4.18
minor sevenths	0.33 ± 0.03	$\mathbf{1.86 \pm 0.90}$	2.95
major sevenths	0.17 ± 0.01	0.70 ± 0.97	2.62
octave jumps	0.49 ± 0.06	0.73 ± 0.23	1.08
total large intervals	6.41 ± 0.20	$\mathbf{18.91 \pm 5.03}$	26.78
percentage of large intervals	24.78 ± 0.52 %	$\mathbf{67.16 \pm 7.45}$ **%**	78.71 %

Table 6.17: Statistics regarding number of intervals in compositions

Name of statistic	**Pre-RL**	**Post-RL**	**Post-RL**
Percentage of leaps resolved	75.18 ± 0.96 %	64.37 ± 6.27 %	73.24 %
Number of leaps resolved	0.73 ± 0.02	$\mathbf{1.58 \pm 0.44}$	**2.02**

Table 6.18: Statistics regarding leaps larger than a fifth per composition

VI **Avoid high correlation** - We can see a noticeable increase in the correlations when compared to section A. This is likely due to the fact that section B features longer notes, which increase the number of held note events in the composition. As we mentioned earlier, this can lead to a high correlation between current and previous compositions, but this is acceptable and is not indicative of repetitive compositions.

Name of statistic	Pre-RL	Post-RL	Post-RL
Correlation with lag 1	0.18 ± 0.01	0.45 ± 0.13	0.26
Correlation with lag 2	0.17 ± 0.02	0.34 ± 0.21	0.13
Correlation with lag 3	0.16 ± 0.01	0.45 ± 0.13	0.61

Table 6.19: Statistics regarding correlation in compositions

VIII **End with a quarter note** - In this section, this rule is much more effective. This is due to the fact that quarter notes are more prevalent given the note length rule, making it easier to learn to compose quarter notes at specific time-steps.

Pre-RL	Post-RL	Post-RL
0.77 ± 0.27 %	**27.66** $\pm$ **19.71** %	**61.10** %

Table 6.20: Statistics regarding the average number of compositions which end with a quarter note

XII **Direction changes** - This rule did not work well in this rule-set. The reason for this is hard to distinguish, but this result is most likely still related to the increased distance between notes on average.

Pre-RL	Post-RL	Post-RL
2.67 ± 0.13	1.90 ± 1.25	**4.101**

Table 6.21: Statistics regarding the average number of changes in pitch direction after three or more of the last six intervals in the same direction per composition.

XIV **End with a quarter note and two quarter rests** - While the first part of this rule worked relatively well, achieving a massive increase in frequency, the second part did not. This is due to the fact that the reward for the pause is only applied when the composition is of the form [...,1,0,0,0,...], whereas the other rewards referring to quarter notes can be applicable in any situation of the form [...,X,0,0,0,...], which is 36 times more likely to appear in the exploration process. This makes successful experiences with late pauses very rare, which leads to an agent that has not learned this behavior properly.

Name of statistic	Pre-RL	Post-RL	Best value
Number of quarter notes at the start of the last bar (per composition)	0.001 ± 0.001	$\mathbf{0.295 \pm 0.196}$	**0.38**
Number of quarter rests in the last 8 notes of the last bar (per composition)	0.001 ± 0.001	0.024 ± 0.028	0.07

Table 6.22: Statistics regarding the average behavior in the last bar of the compositions.

6.4 Other RL configurations

6.4.1 Other RNNs with both rule-sets

Galician Basic RNN (2.5)

As mentioned earlier, the other RNNs were trained with the same rule-sets, using the same values for rewards and the same hyper-parameters. The Galician Basic RNN at 85 % accuracy reacted well to the Reinforcement Learning training, mantaining its musicality and composing varied and somewhat pleasing pieces. The note repetition was cut down significantly to only 0.4 % and the notes off key were also reduced to approximately 1 %. The note length rules worked well, apart from a lack of sixteenth notes in section A, since the Galician rule-set was tuned for the pre-trained version of the network, which composes many sixteenth notes. Some other rules such as promoting the compositions of thirds or early pauses did not manage to affect the outputs consistently, but with some manual tuning (specifically for this configuration) this can be solved. For higher accuracies, the network clearly becomes less flexible and composes with adequate statistics but lacking any musicality, focusing on two or three notes for the entirety of the piece. This demonstrates how important it is to monitor the accuracy of an RNN when combining it with the RL Tuner.

Pre-trained+Galician Basic RNN (3.5)

The Pre-trained+Galician Basic RNN performed virtually as well as the Pre-trained RNN with both rule-sets, which is remarkable. For every accuracy, the network displayed no excessive repetition, few notes off key, appropriate intervals and note lengths, and even managed to fulfill some of the more demanding tasks such as composing early pauses in 99.8 % of compositions in our training run. The pieces themselves also feature a lot of variety while still sounding very Galician due to frequent ascensions and descents in pitch. *The fact that this network worked so well is a testament to the fact that composing good music should always involve a mix of a general and specific dataset, which is the case for this RNN configuration and rule-set.*

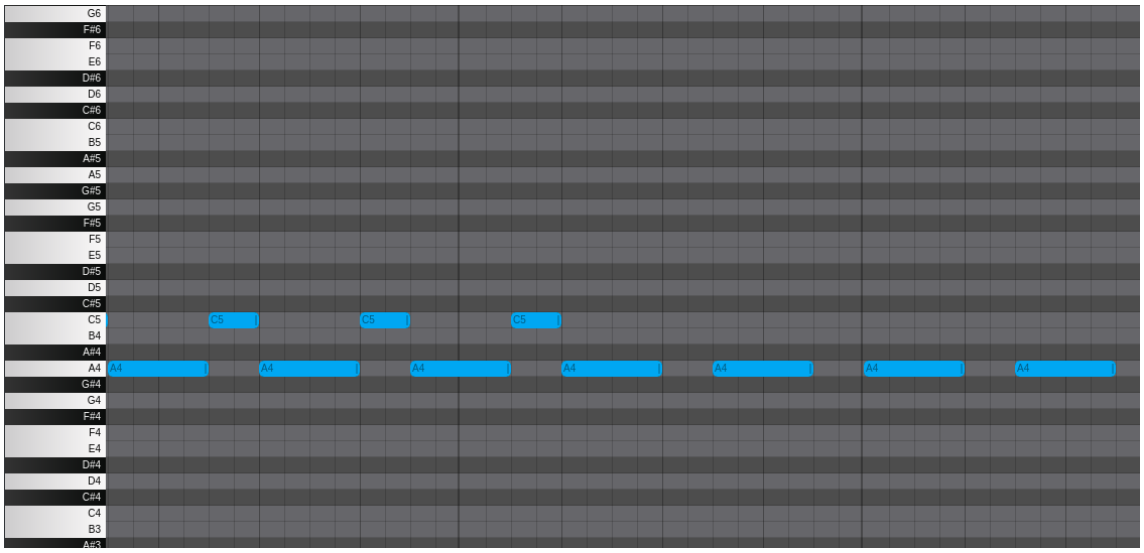


Figure 6.3: This image is taken from the piano-roll representation of a sample generated by the Galician Basic RNN trained with Both Rule-sets (2.5). We can clearly see that the network is being repetitive but this behavior would not be punished by the auto-correlation rule, given the distance between notes, or by the repetition rule since it does not repeat the same note more than eight times.

Galician Attention RNN (4.5)

Unfortunately, the Galician Attention RNN was not as successful as the others. It appears that the network did not react well to the RL Training for any of the accuracies, performing poorly in many measures. The notes off key, for instance, are very prevalent (between 20 % and 30 % of all notes), and the excessively repeated notes reached 6.7 % in one of the training runs, which is unacceptable and evidently noticeable. The outputs display these issues clearly, as well as a lack of consistency in note lengths throughout a single piece and jarring variations in tonal range. The reason for this poor performance is difficult to pinpoint, especially since care was taken to monitor the network's accuracy in order to prevent it from overfitting. It is possible that due to the large amount of training steps taken when compared to the other RNNs, this network becomes too confident about the next note and does not adapt to the rule-set, but the note density probabilities do not indicate this clearly.

6.4.2 Other rule-sets

Zero reward(X.2)

[3]The only purpose of experimenting with this reward mode is to display a fundamental difference between the RNN and the RL Tuner environment. One would expect that this reward mode, containing only the reward from the reward RNN, would yield equivalent results to the original RNN. However, we must recall that the network attempts to learn which behavior leads to the most reward. Since usually

[3]X.2 Refers to configuration 1.2, 2.2, 3.2, 4.2 and 5.2 .

the most likely note is the held note event, the network starts composing using this action at virtually every state in order to maximize the reward. This means that the final compositions have a very noticeable excess of long notes and pauses, sometimes longer than 16 time-steps. We can conclude that the role of the rule-set as part of the reward function is to steer the composition away from this behavior, rewarding relevant behavior and not just absence of notes. In other words, the RL environment requires a well-rounded rule-set in order not to descend into this passive mode of composition, which is an important aspect of this architecture not highlighted in the original paper [2].

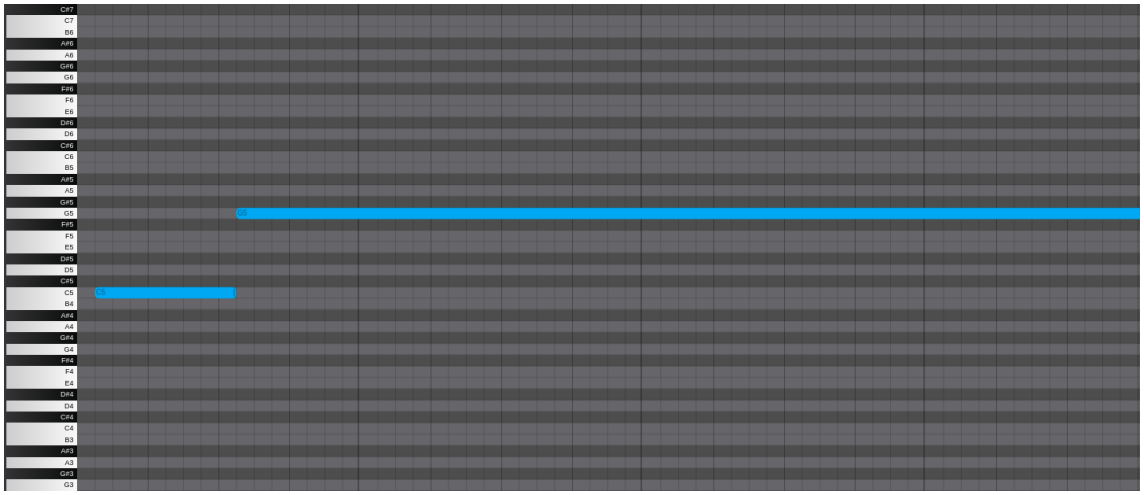


Figure 6.4: This image is taken from the piano-roll representation of a sample generated by the Pre-trained Basic RNN trained with the Zero Reward mode (3.2). We can see that the composition features a very excessive amount of held note events (0, in the RL Tuner representation).

Original rule-set(X.3)

The original rule-set achieved success with the pre-trained RNNs it was designed for. The pre-trained RNN as well as the Pre-trained+Galician Basic RNN benefited from the rule-set by repeating notes less often and staying in key, as well as avoiding long notes or silences, as was expected. However, it is worth noting that this rule-set did not work well for the other two RNNs, failing to achieve even its most essential functions such as avoiding repetition. The exception to this phenomenon was the Galician Basic RNN with 85 % accuracy, which performed acceptably but not particularly well. The reason for this behavior is perhaps that the original rule-set features relatively low values for its rewards and did not manage to influence the final compositions. The idea behind trying this rule-set with every configuration was to show empirically that it does not work well every time and it is not viable for any application (for specific genres, it is better to use specific rule-sets).

Galician rule-set(X.4)

This reward mode was surprisingly effective with some configurations. Namely, the Pre-trained+Galician Basic RNN and the Galician Basic RNN benefited immensely from these rewards, and did not have an issue with key adherence or repetition as would be expected with a rule-set which does not help prevent these issues. Instead, the compositions achieved the intended statistics and were in fact comparable to the main configuration. With the pre-trained RNN, however, this was not the case, leading to plenty of repetition and displeasing note choices. This is indicative of the issues that the pre-trained RNN has when compared to the other RNNs trained with a more specific dataset. The attention RNN also did not work well with this rule-set, although it mostly managed to stay in key without any associated reward for this purpose, which is remarkable.

6.5 Summary of the results

Given the amount of statistics which are relevant to evaluate the results, as well as the variety of RNN/rule-set configurations tested, it is extremely challenging to summarize the analysis of the experimental procedure. With this in mind, we decided to elaborate a comprehensive set of charts which highlight the effectiveness of each reward for every configuration. Specifically, these charts represent heat maps which measure the variation of a statistic after the RL training. This variation is taken as positive or negative, depending on the behavior we would like to observe in the section. For instance, in section B, an increase of repeated note by 15 % would be added to be observed as -15 in the graph, while an increase of quarter notes by 10 would be seen as +10. These variations are then summed into a total variation which acts as a rating of how well the configuration adapted to the rule-set. It is worth noting that these heat maps are useful as summaries of the results, but they do not encompass all of the complexities which were discussed in the previous sections, which is why they were not presented earlier.

By visualizing the statistics in this manner, we can get a general idea of the effectiveness of each configuration, which is extremely useful as a summarization of the results that have been presented in this chapter. By ranking the configurations according to their rating (total variation), we can see that these charts are in line with what was mentioned before. The pre-trained RNNs typically are more successful due to their malleability, as mentioned before. The combination of the rule-sets is the most effective, since it the reward values were tuned for it, but the Galician rule-set on itself is also quite effective in some cases. If we briefly look at the heat maps for configurations 4.X, it is very clear that the Galician Attention RNN does not yield good results, as we mentioned above.

Legend for the heat maps:

- The value of each cell is determined by the variation of each statistic, as explained above. Positive variations are seen in blue, whereas negative variations

#	Section A	Total variation	Section B	Total variation
1	1.5	223.36 (avg.)	1.5	219.45 (avg.)
2	3.5 (0.95)	210.13	2.4 (0.95)	195.15
3	3.4 (0.9)	203.00	1.4	156.12
4	3.4 (0.95)	183.03	3.5 (0.85)	149.08
5	3.4 (0.85)	177.00	3.4 (0.85)	128.93

Table 6.23: This table ranks the configurations according to their total variation in the heatmaps below.

are seen in red, as detailed in the legend on the right of each chart.

- The rows represent the configurations that were tested (RNN/rule-set combinations), which we denominated in section 6.1. Configuration 1.5 was the main object of our study, and therefore its training procedure was run five separate times, hence the configurations of the form " 1.5 (run X) ". The same is the case for the other RNNs, which were trained for three different training accuracies, indicated by "(0.85)", "(0.9)" or "(0.95)".
- The columns represent the number of the statistic, which corresponds to the number of the line in which it appears in the original output file. The legend for the statistics (columns) can be seen below:

#	**Name of statistic**
17	Percentage of notes off key
18	Percentage of notes in motif
19	Percentage of notes in repeated motif
20	Percentage of excessively repeated notes
22	Avg. autocorrelation with lag 1
23	Avg. autocorrelation with lag 2
24	Avg. autocorrelation with lag 3
27	Avg. num. of major sevenths per composition
28	Avg. num. of (non-preferred) perfect fifths per composition
29	Avg. num. of major sixths per composition
30	Avg. num. of perfect fourths per composition
33	Avg. num. of (non-preferred) major thirds per composition
34	Avg. num. of preferred perfect fifths per composition
40	Avg. num. of larger notes per composition
42	Avg. num. of whole notes per composition
44	Avg. num. of half notes per composition
46	Avg. num. of quarter notes per composition
48	Avg. num. of eighth notes per composition
50	Avg. num. of sixteenth notes per composition
52	Percent. of compositions starting with a quarter rest
54	Percent. of compositions starting the last bar with quarter note
56	Percent. of compositions ending the last bar with one/two quarter rests [4]
58	Percent. of compositions ending with a quarter note)
60	Percent. of composition ending with the tonic
62	Avg. num. of sub-tonic/tonic sequences per composition
64	Avg. num. of opposite seconds after leap per composition
66	Avg. num. of direction changes per composition

Table 6.24: This table ranks the configurations according to their total variation in the heat maps below.

[4] Using this measure, compositions that end with two quarter note rests are counted twice for the final percentage. This is acceptable for the heat maps since this percentage is still a good representation of the the effectiveness of the rule. However, this measure is not as rigorous as the one presented in section 6.3.

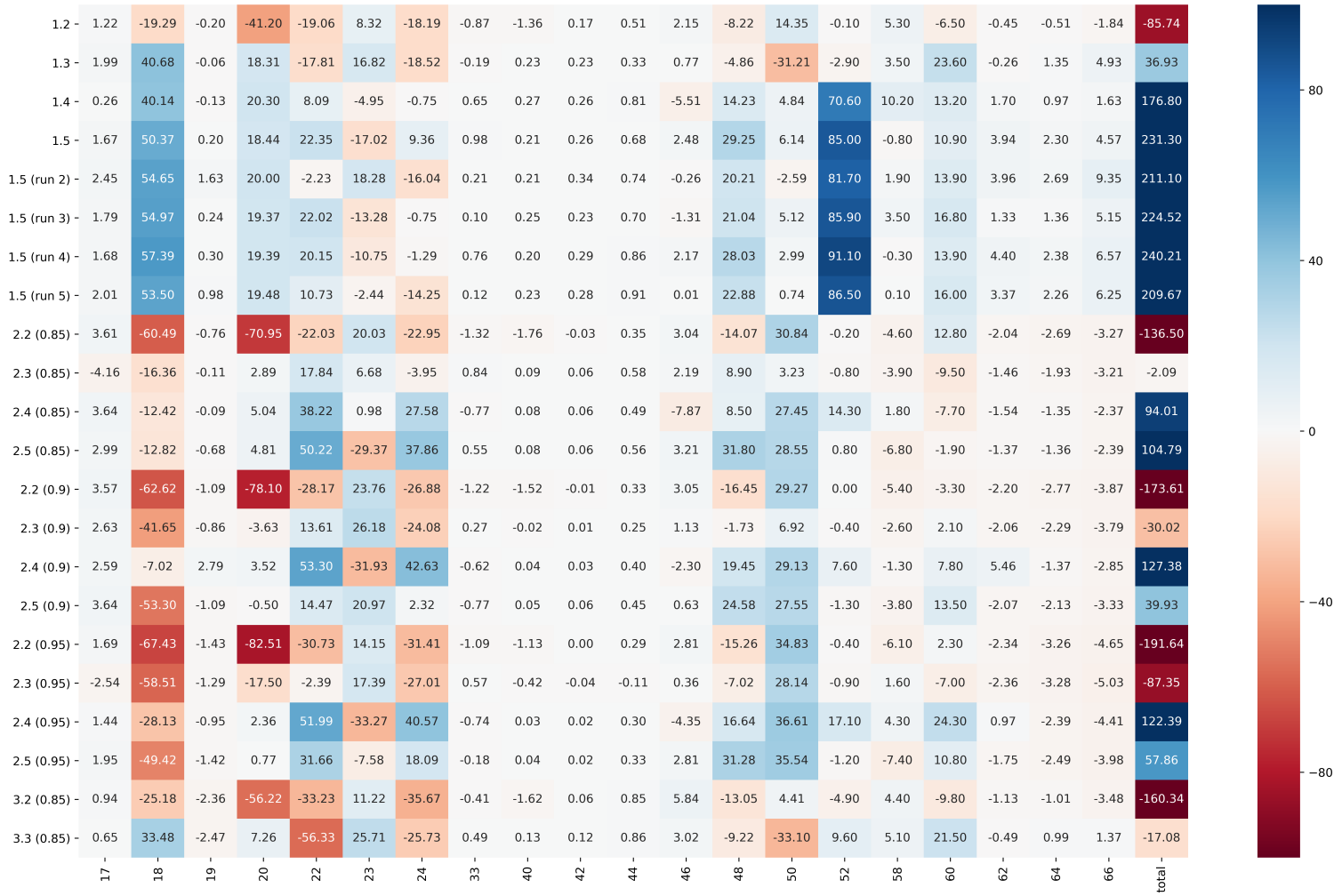


Figure 6.5: Statistical variations between pre and post-RL compositions, Section A, part 1

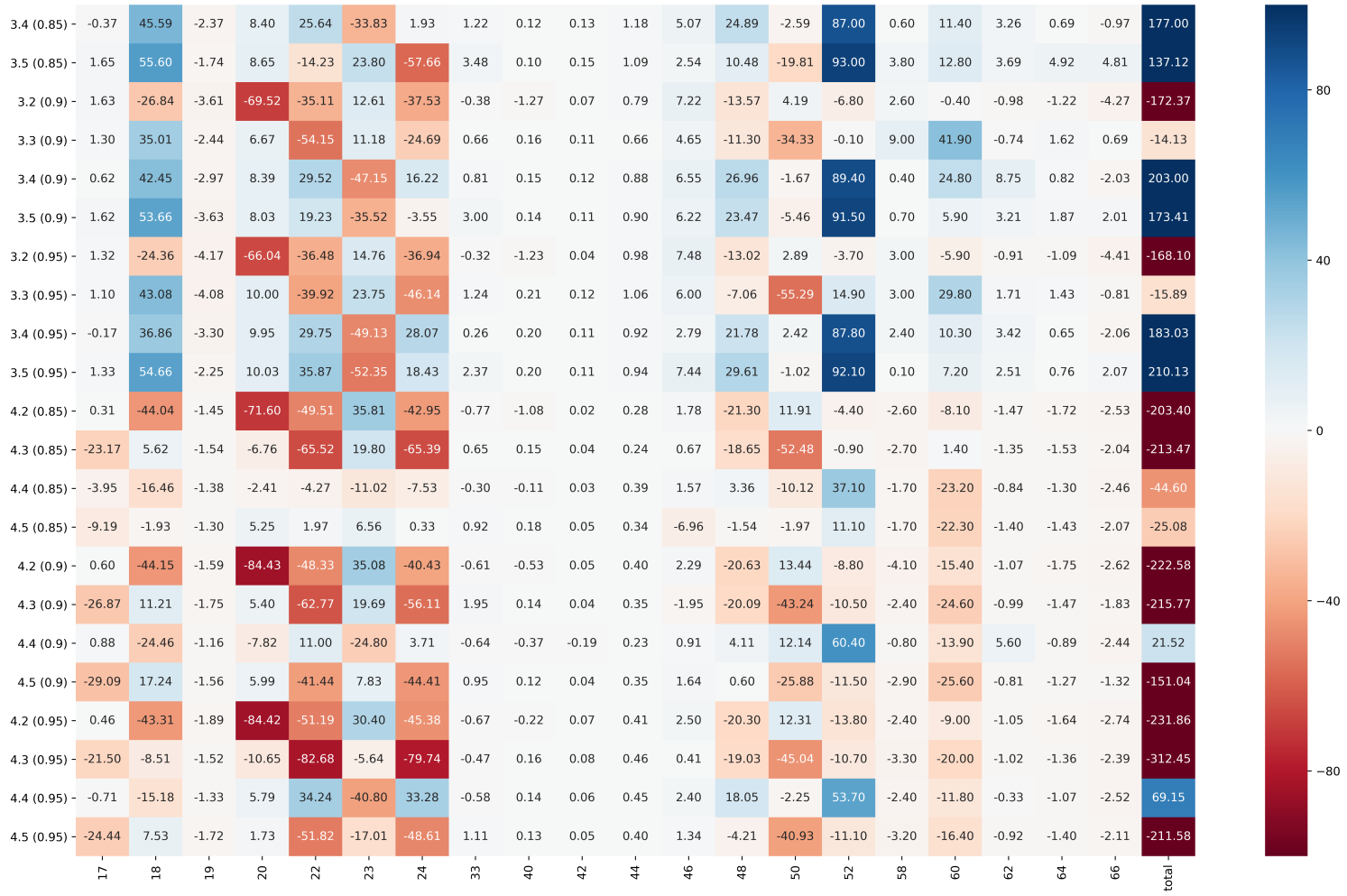


Figure 6.6: Statistical variations between pre and post-RL compositions, Section A, part 2

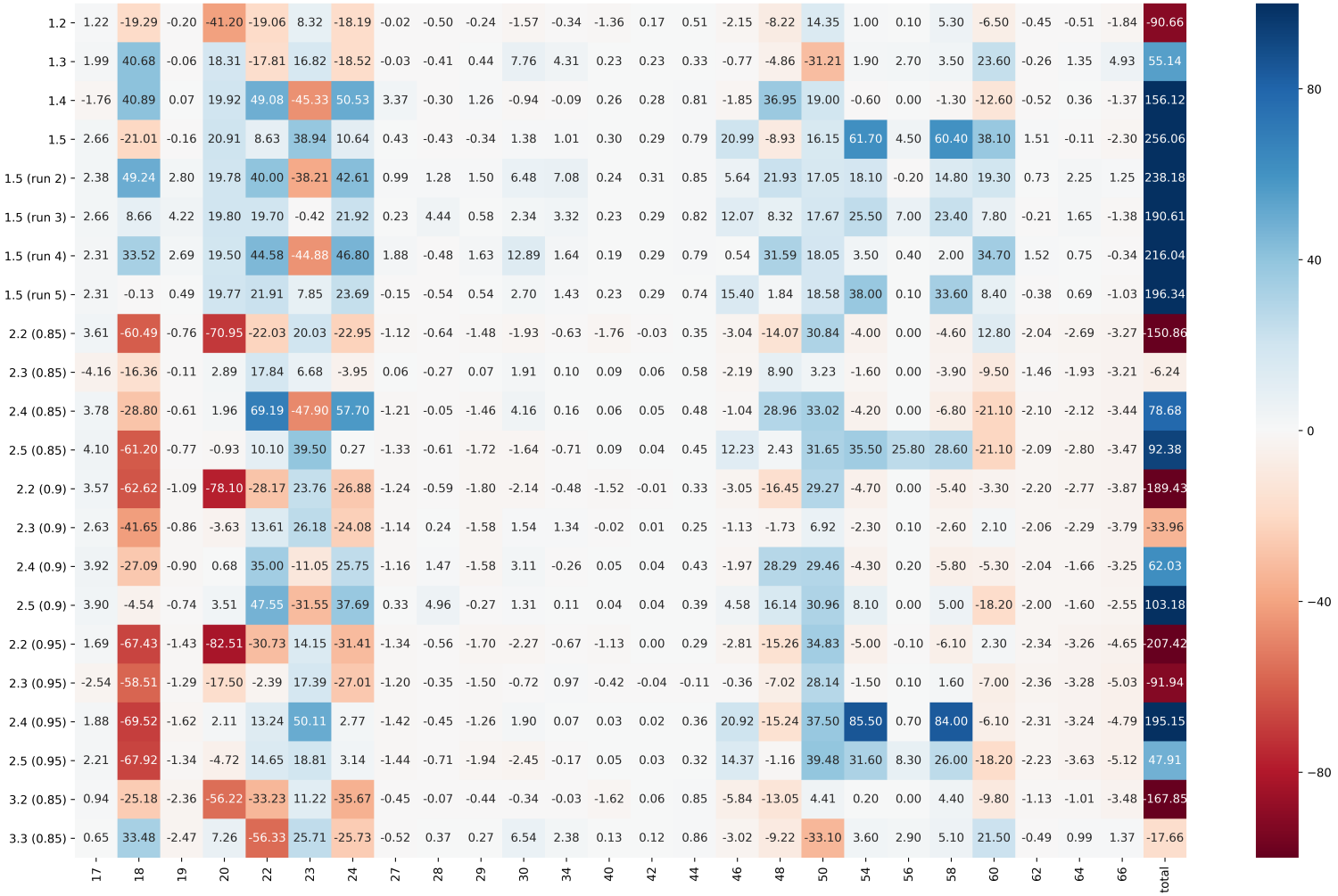


Figure 6.7: Statistical variations between pre and post-RL compositions, Section B, part 1

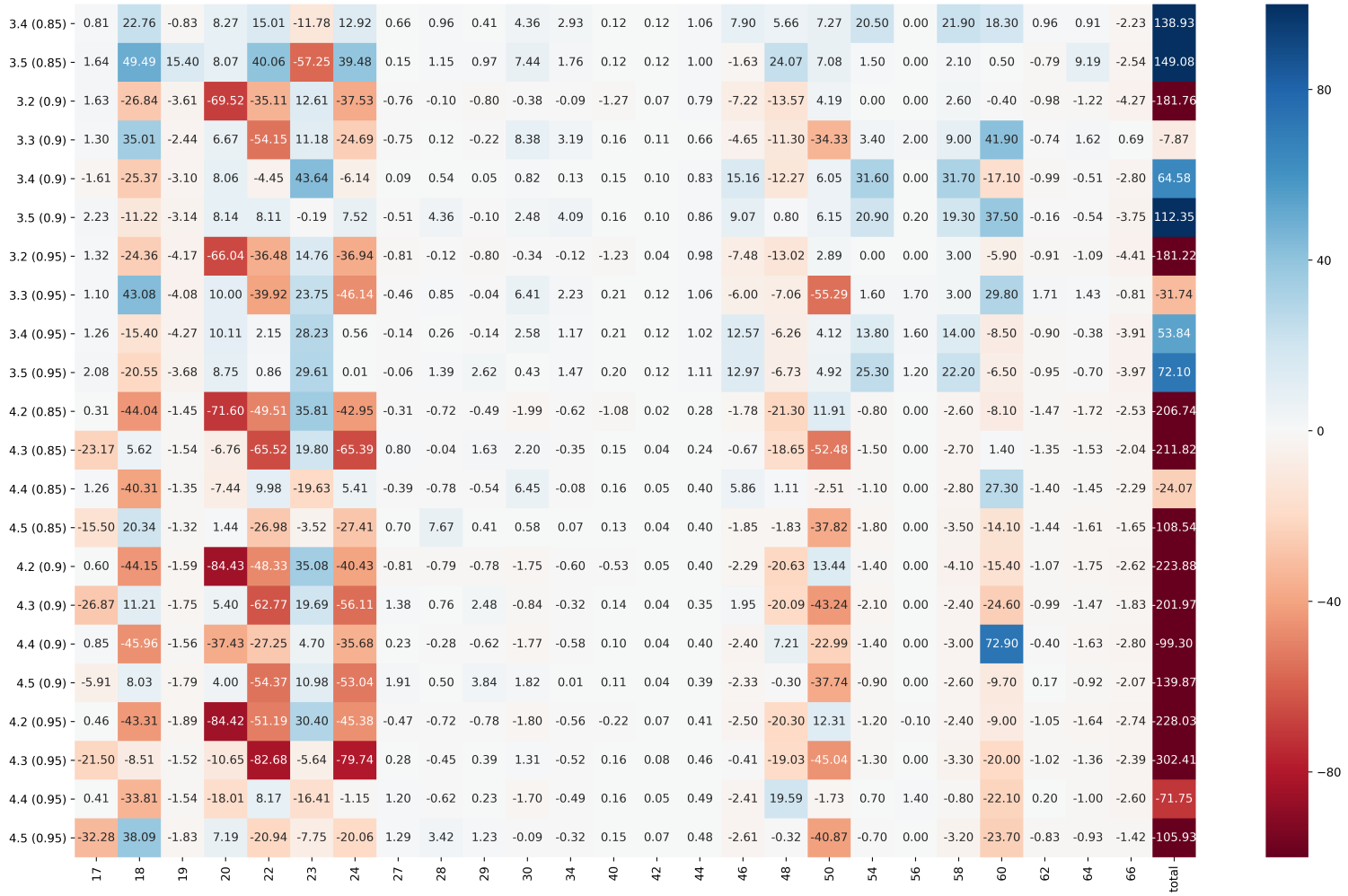


Figure 6.8: Statistical variations between pre and post-RL compositions, Section B, part 2

Chapter 7

Conclusions and future work

In this chapter we conclude the report by summarizing what we have observed in the previous chapters and suggesting some future work. The first section is dedicated to summarizing what has been mentioned in the earlier sections, explaining the main objective of this project and how it was achieved, focusing on the contributions and the original work done. Then we summarize the findings that were observed after analyzing the experimental results. This is one of the most relevant parts of the report and contains essential information for any future authors using the RL Tuner for research. We finish by suggesting two extensions to this work regarding Inverse Reinforcement Learning (IRL) and emotional rule-sets (happy or sad), pairing them with two respective proofs of concept to motivate further research.

7.1 Overview of the project

In this report, our goal was to compose in a specific style using the RL Tuner and the Melody RNN. First, we chose a simple genre - *The Galician Xota* - and gathered a dataset including only pieces from this genre. We then adapted the dataset by converting it to be fully MIDI based and separating tracks in order to have polyphonic files only. After this, we augmented the dataset adapting a known method and trained three different RNN configurations with our dataset: the Basic RNN, the pre-trained Basic RNN and the New Attention RNN, the latter of which we created by adding attention mechanisms to the Basic RNN using the existing Melody RNN code. We monitored these training attempts, recording their training and evaluation accuracy in order to produce checkpoints which were not overfitting our dataset.

After this, we created a rule-set which represented our genre. This involved manually analyzing the dataset to infer the underlying musical composition rules such as frequency of certain intervals and typical note lengths, as well as some more specific behavior. The new rule-set was divided into two different sets, one for each type of section in the piece, which share some of the same rules but differ in others. We then translated these conceptual rule-sets into a set of rewards to be used as the music theory reward in the RL Tuner, and tuned these values using the Pre-trained RNN with the combination of the new and original rule-sets. This led to four new reward

modes, the Galician rule-set (Section A and B) and the combination of the Galician and original rule-sets (Section A and B).

After this, experiments were performed for every combination of the Pre-trained RNN, Galician Basic RNN, Pre-trained+Galician Basic RNN and the Galician Attention RNN with the five different reward modes available: no_rl (no RL training), zero_reward (no music theory reward), music_theory_only (original rule-set), Galician_only (new rule-set only) and both_rule_sets (original+new rule-set). The main configuration (Pre-trained RNN, both Rule-sets) was analyzed in detail and the other configurations' outputs were also taken into account to produce a detailed set of findings about the RL Tuner training procedure and the development of new rewards, suggesting how it should be done in future work.

7.2 Summary of novel findings

The extensive analysis of every configuration as well as the other training procedures mentioned in previous chapters highlight a remarkable set of findings regarding the RL Tuner and artificial music composition in general:

1. The impact of a music theory reward is influenced by the following factors:

    - **Position in time** - Rewarding early behaviors is generally easier than late behaviors because the possible state space increases with time (at $t = 1$ there are 38 possible states, at $t = 2$ there are 38^2). This is supported by the success of the rule regarding early pauses compared to the failure of the rule regarding late pauses.
    - **Event specificity** - Rewarding quarter notes is a lot more effective than rewarding quarter rests since the first reward spans 37 events and is therefore much more likely to appear in exploration and allow the network to learn this behavior. We can observe this behavior in rule XIV.
    - **Time specificity** - Rewarding a behavior generally (without specifying a time in the composition) is a lot more effective than rewarding a behavior at a certain time-step only. This can be seen by comparing the results of rules IV and VIII.
    - **Complexity of behavior** - During initial experiments, some more complex rules were tried. However, the Deep Q Network had difficulty learning these rules since the relation between the state/action and the reward was not evident or consistent.
    - **Distance between action and reward** - The late tonic rule worked relatively well but it did not manage to influence the majority of compositions. This is likely due to the fact that the reward for finishing with this note is attributed at the end of the composition, which is usually some time-steps away from the state at which this action was taken, which alienates the reward attribution from the action that caused it. This requires the state

to model previous notes very accurately so that the Deep Q Network can understand why the reward was attributed, but it is often unreliable.

- **Sparsity** - Sparse rewards were inevitable in this project, as they are in many other real world scenarios, but they are generally problematic when using traditional Deep Q Learning. In some cases, we would like to reward the network for composing a quarter rest ([1,0,0,0]), but this is a difficult behavior to learn since the reward is only attributed when this exact sequence occurs and not when a similar one occurs, such as [1,0,0,2]. This issue is very prevalent in the Reinforcement Learning community and there are some solutions which could be useful [56], but they would require substantial changes to the architecture.
- **Oppression** - The two discarded rules mentioned in 5.2.2 clearly show the role that an oppressive rule can play in an RL environment. The fact is that some rules/rewards can overpower the agent and negatively affect the impact of other rules. This issue can sometimes be alleviated by tuning the value of the reward function, but sometimes the concept of the reward itself is not appropriate. In our environment, any rewards that promote repetition, notes off key or long silences are generally considered oppressive because they tend to cause the network to degenerate into a model which ignores other rules and loses its musicality.
- **Value of the reward** - This is the most obvious of the factors. Evidently, attributing a larger reward to a certain behavior will make it more prevalent in the post-RL compositions. However, it is important to tune the reward values in order to find a balance between the rules, and larger rewards may not always lead to better results. Additionally, the reward scaler must be taken into account when choosing these values, since acts as a multiplier for the total music theory reward.

2. Every parameter of the RL Tuner, including the number of training steps and the RL algorithm, can directly influence the effectiveness of a set of music theory rewards.

3. Intra-procedural and inter-procedural variance (as defined in 5.3) are serious issues, which should be taken into account in order to produce meaningful statistics about post-RL results.

4. Using a trained recurrent neural network with very high training accuracy can make for a highly inflexible Reward RNN which does not adapt well to music theory rewards since its note probability density is very narrow and does not allow for other viable actions. To prevent this, the training accuracy must be monitored during the RNN training procedure and checkpoints must be produced at earlier stages, before the accuracy approaches 100 %. In other words, it is necessary not only to prevent overfitting, but overconfidence as well.

5. Adding attention mechanisms to an RNN requires more training steps and does not necessarily improve the outputs in the context of music composition. This

means that a lot of the credit for the impressive outputs of the original Attention RNN [6] must be attributed to the lookback encoding. The RL Tuner also does not work well with attention mechanisms in our experience, but with the lookback encoding or some other augmentation this model may be more compatible with the RL rewards.

6. Networks trained with large, varied datasets tend to produce a fair amount of underwhelming results but remain flexible, which is ideal for the RL Tuner. This is the case even after being trained with a smaller dataset, as is the case for the Pre-trained+Galician Basic RNN.

7. The quality of pre-RL and post-RL compositions are not necessarily directly correlated.

8. The issues of excessive repetition and overly long notes are not equally serious for every network. The Galician Basic RNN, for instance, did not suffer from these issues as much as the other networks.

9. The original rule-set is not always effective at its most basic tasks such as avoiding repetition, as we saw with the Galician Basic RNN and the New Attention RNN.

10. A proper music theory reward is essential and without it the RL Tuner can deteriorate the quality of the compositions, as we saw with the zero reward mode.

7.3 Future work with proofs of concept

7.3.1 Inverse Reinforcement Learning

There are two major extensions which can be done to this work. The first of these is to automate the attribution of specific values to rewards. The fact is that tuning the rewards was the most time consuming part of this project and it is clear that for the original RL Tuner this tuning was performed using grid search (attempting every combination with a set of possible values for each reward), which surely required a very large amount of resources. To be clear, each of our training runs took approximately two hours on an Nvidia Geforce GTX 1080. Ideally, it would be interesting to apply a method that is used in areas such as Robotics, commonly know as Inverse Reinforcement Learning (IRL). This could be implemented in different manners, but entropy based methods such as [57] have been very influential and would possibly be a good fit. This would be a great breakthrough since it would automate the bottleneck of developing a new rule-set and therefore improve the style-transferring capability of the RL Tuner.

To prove that this idea is indeed viable, we decided to perform a brief proof of concept. The objective was to make the Pre-trained Basic RNN compose as many eighth notes (or more) has we had seen by analyzing the dataset. First, we had

to adapt the code which is used to measure statistics of pre/post-RL compositions in real time (as they are being composed), in order to measure the same statistics on MIDI files. After this was performed, we measured the amount of eighth notes which would be in a Galician composition if it were 96 notes long (18.26) and set this as the target for our network. Then, we begin with a zero reward for composing the eighth note and check the post-RL statistics. If it is below the target, we run the training procedure again, increasing the reward value by 5. After 3 iterations, the network produced an adequate number of eighth notes per composition (33.65), and the procedure was complete. This is evidently a very simple and inefficient way to perform IRL, but it empirically demonstrates that transferring the style of a MIDI dataset or RNN composer to another RNN composer using the RL Tuner is possible. The results/code for this experiment can be obtained from the project's repository [5].

7.3.2 Emotional rule-set

The second extension which could be applied to the current work would be an emotional rule-set. Creating music which provokes specific emotions is a very powerful idea since this could be the breakthrough that artificial music composition needs to be used by the general public. In this project we have demonstrated that the RL Tuner can be used not only to tune an RNN with some general rules, but can also be used to drastically change its compositional style. The main aspects which determine the emotion of a song are its scale, frequent intervals and rhythmic characteristics [58, 59]. These can all be tuned using the RL Tuner, which means that the development of an emotional rule-set could allow us to steer an RNN composer based on any dataset to a specific emotion, accomplishing the task of controlled affective musical composition.

Again, we decided to elaborate a small demonstration in order to show that this concept is viable. We implemented two different reward functions: one for happy, joyful music and the other one for sad, melancholic music. The first was made to compose in C Major, rewarding shorter, major intervals and faster notes. The second was made to compose in C Minor, rewarding minor intervals and slower notes, and encouraging rests (note off events). We also included some of the rewards from the original rule-set in for both emotions. After training the Pre-trained Basic RNN with these two rule-sets, some surprisingly successful results were obtained. Specifically, the compositions reacted well to these rule-sets, composing music which respects the rules but is still diverse. In the end, we believe it would be easy to distinguish the happy compositions from the sad ones and classify them correctly, which is extremely promising. The samples and statistics for this experiment can be found in the project repository [5].

Appendices

Appendix A

Relevant code from Magenta

This code was taken directly from the Magenta repository [55] in order to ensure that it had not been modified.

```
165        'attention_rnn': MelodyRnnConfig(
166            magenta.protobuf.generator_pb2.GeneratorDetails(
167                id='attention_rnn',
168                description='Melody RNN with lookback encoding and attention.'),
169            magenta.music.KeyMelodyEncoderDecoder(
170                min_note=DEFAULT_MIN_NOTE,
171                max_note=DEFAULT_MAX_NOTE),
172            tf.contrib.training.HParams(
173                batch_size=128,
174                rnn_layer_sizes=[128, 128],
175                dropout_keep_prob=0.5,
176                attn_length=40,
177                clip_norm=3,
178                learning_rate=0.001))
```

Figure A.1: This picture is taken from the Melody RNN code, and it clearly shows that the original Attention RNN used the lookback encoding.

```
125  def autocorrelate(signal, lag=1):
126    """Gives the correlation coefficient for the signal's correlation with itself.
127
128    Args:
129      signal: The signal on which to compute the autocorrelation. Can be a list.
130      lag: The offset at which to correlate the signal with itself. E.g. if lag
131        is 1, will compute the correlation between the signal and itself 1 beat
132        later.
133    Returns:
134      Correlation coefficient.
135    """
136    n = len(signal)
137    x = np.asarray(signal) - np.mean(signal)
138    c0 = np.var(signal)
139
140    return (x[lag:] * x[:n - lag]).sum() / float(n) / c0
```

Figure A.2: This picture is taken from the original RL Tuner code, and it displays the function that computes the autocorrelation for rule 7 (mentioned in 5.2.1)

Appendix B

RNN Training Charts

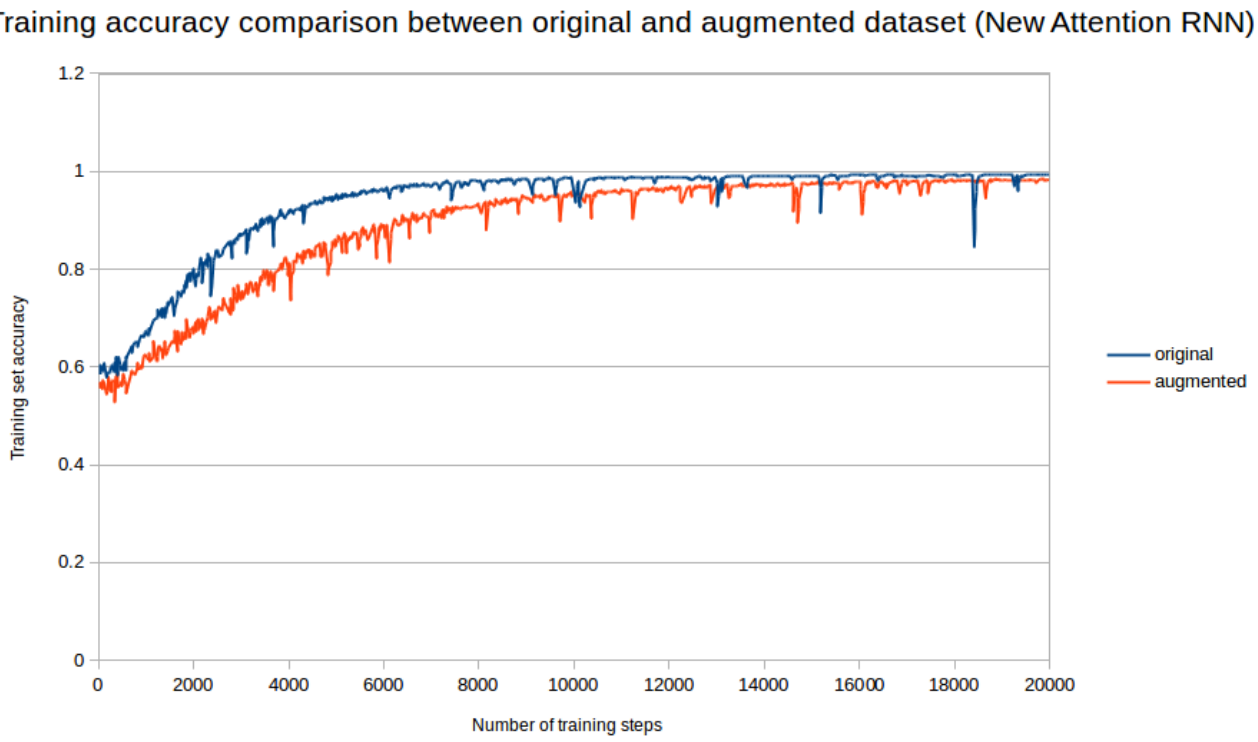


Figure B.1: Chart comparing the training accuracy of the Basic RNN with the original (non augmented) dataset and the new augmented dataset.

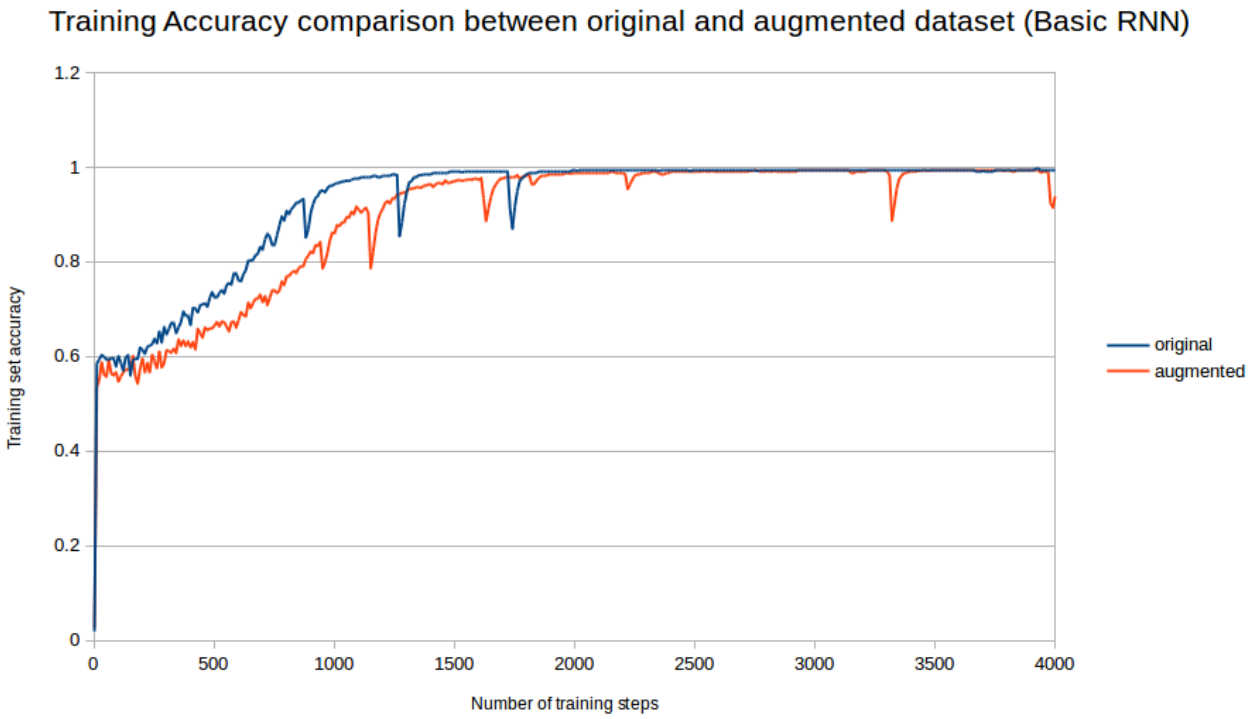


Figure B.2: Chart comparing the training accuracy of the New Attention RNN with the original (non augmented) dataset and the new augmented dataset.

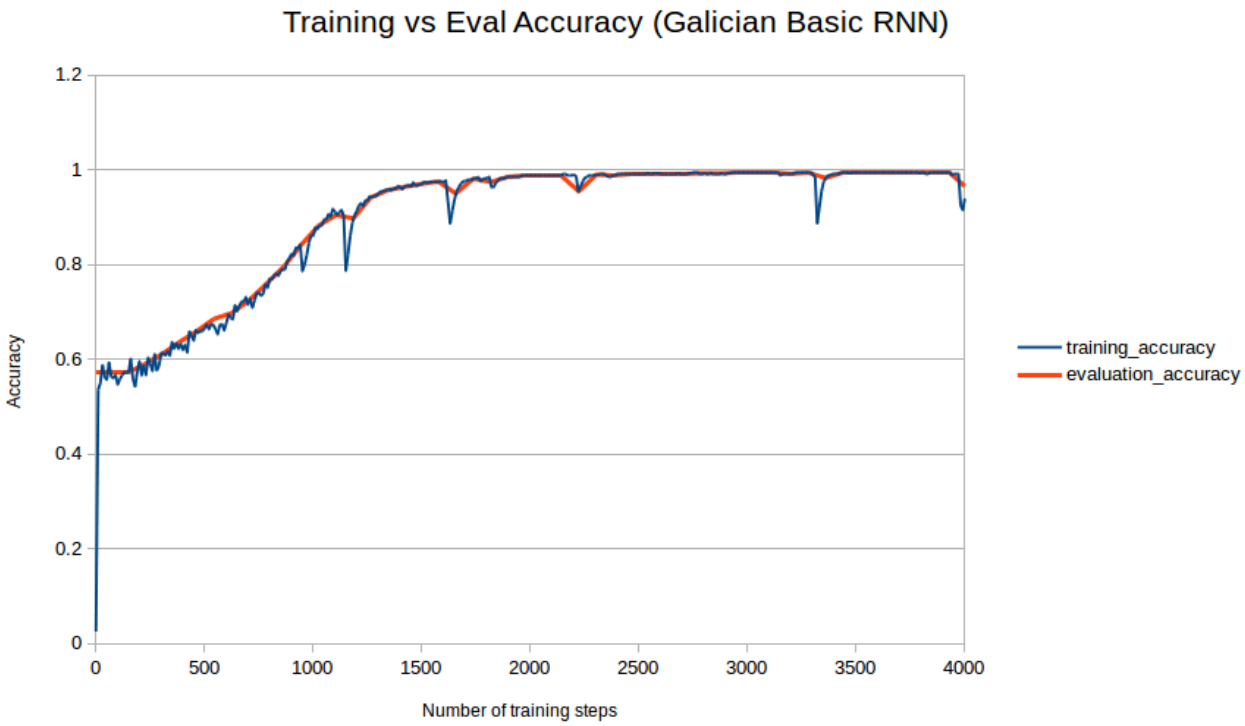


Figure B.3: Chart portraying the training accuracy and evaluation accuracy throughout the Galician Basic RNN training procedure.

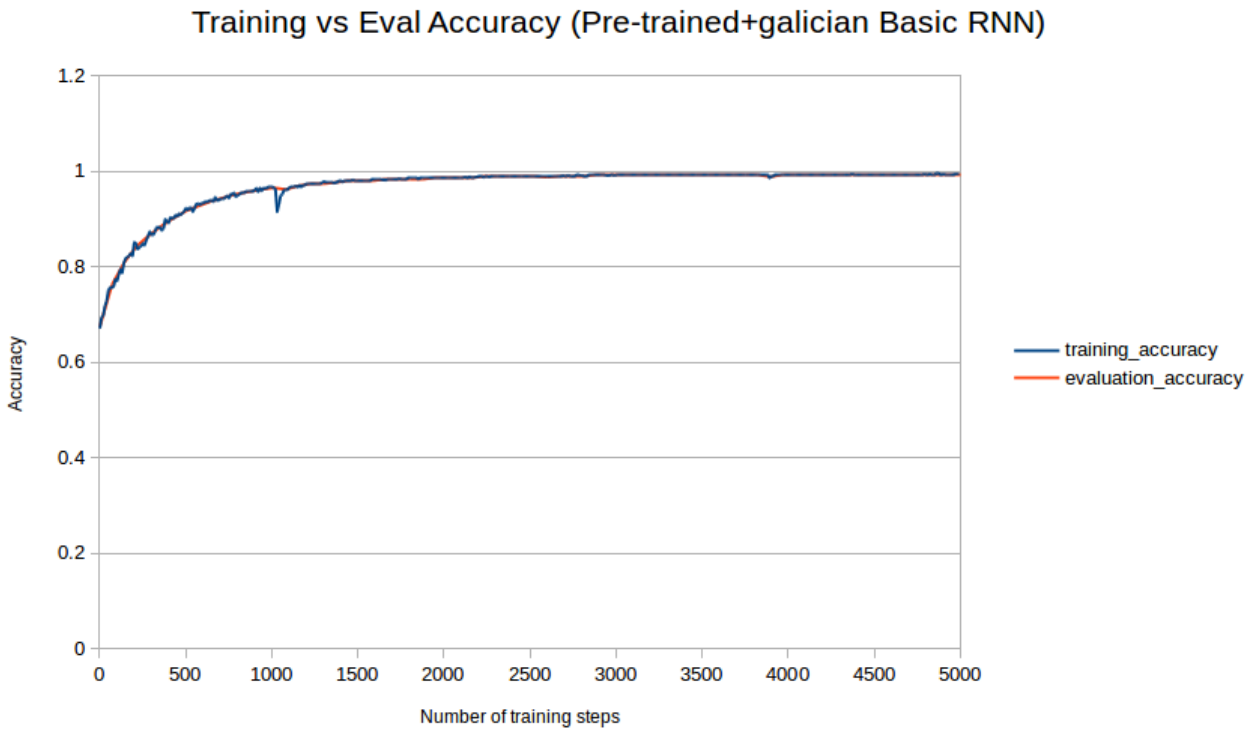


Figure B.4: Chart portraying the training accuracy and evaluation accuracy throughout the Pre-trained+Galician Basic RNN training procedure.

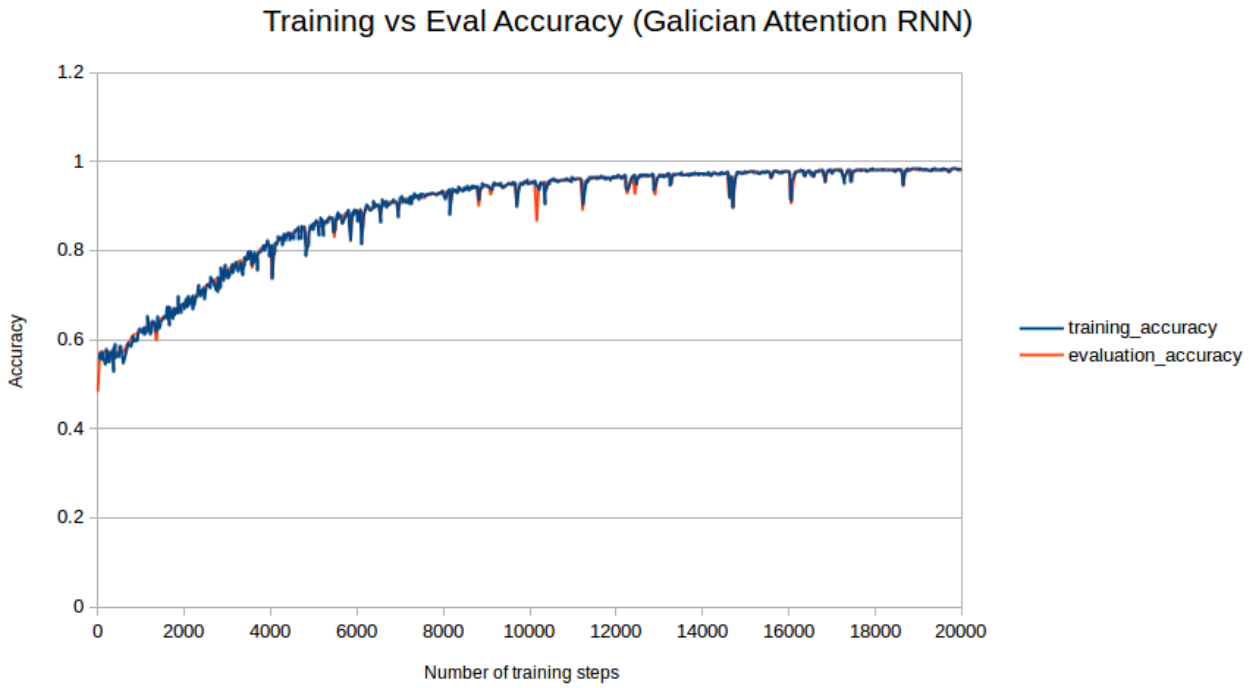


Figure B.5: Chart portraying the training accuracy and evaluation accuracy throughout the Galician Attention RNN training procedure.

Appendix C

Ethics checklist

	Yes	No
Section 1: HUMAN EMBRYOS/FOETUSES		
Does your project involve Human Embryonic Stem Cells?		✓
Does your project involve the use of human embryos?		✓
Does your project involve the use of human foetal tissues / cells?		✓
Section 2: HUMANS		
Does your project involve human participants?		✓
Section 3: HUMAN CELLS / TISSUES		
Does your project involve human cells or tissues? (Other than from Human Embryos/Foetuses i.e. Section 1)?		✓
Section 4: PROTECTION OF PERSONAL DATA		
Does your project involve personal data collection and/or processing?		✓
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?		✓
Does it involve processing of genetic information?		✓
Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.		✓
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?		✓
Section 5: ANIMALS		
Does your project involve animals?		✓
Section 6: DEVELOPING COUNTRIES		
Does your project involve developing countries?		✓
If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?		✓
Could the situation in the country put the individuals taking part in the project at risk?		✓

	Yes	No
Section 7: ENVIRONMENTAL PROTECTION AND SAFETY		
Does your project involve the use of elements that may cause harm to the environment, animals or plants?		✓
Does your project deal with endangered fauna and/or flora /protected areas?		✓
Does your project involve the use of elements that may cause harm to humans, including project staff?		✓
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?		✓
Section 8: DUAL USE		
Does your project have the potential for military applications?		✓
Does your project have an exclusive civilian application focus?		✓
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?		✓
Does your project affect current standards in military ethics e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?		✓
Section 9: MISUSE		
Does your project have the potential for malevolent/criminal/terrorist abuse?		✓
Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?		✓
Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?		✓
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project?		✓
SECTION 10: LEGAL ISSUES		
Will your project use or produce software for which there are copyright licensing implications?		✓
Will your project use or produce goods or information for which there are data protection, or other legal implications?		✓
SECTION 11: OTHER ETHICS ISSUES		
Are there any other ethics issues that should be taken into consideration?		✓

Appendix D

Ethical and professional considerations

Regarding sections 1, 2 and 3, my project does not involve humans, embryos, fetuses, cells or tissues in any way, so the answer was “No” for all of the questions. Regarding section 4, my project does not involve any personal or private data, so the answers were all “No” Regarding section 5, my project does not involve animals as can be clearly seen. Regarding section 6, my project is not related to any developing countries, so the answer was “No’. Regarding question 7, my project cannot possibly have any impact on the environment and definitely does not constitute a safety hazard, so the answers were all “No”. Regarding section 8, my project has no military foreseeable military use, so I answered all questions negatively. Regarding section 9, my project does not have any relevant damaging impacts on mankind or society, so all of the answers were “No”.

Regarding section 10, the musical pieces used for the dataset can potentially have copyright associated with them, which would also imply that the generated compositions could involve copyright issues. However, after consulting all of the sources there was clearly no copyright information, so the answer for both of the questions in this section was “No”. Finally, I believe there are no other ethical issues to discuss regarding this project.

Bibliography

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

[2] Natasha Jaques, Shixiang Gu, Richard E. Turner, and Douglas Eck. Tuning recurrent neural networks with reinforcement learning. *CoRR*, abs/1611.02796, 2016. URL http://arxiv.org/abs/1611.02796.

[3] Olof Mogren. C-RNN-GAN: continuous recurrent neural networks with adversarial training. *CoRR*, abs/1611.09904, 2016. URL http://arxiv.org/abs/1611.09904.

[4] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. In *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017, Suzhou, China, October 23-27, 2017*, pages 324–331, 2017. URL https://ismir2017.smcnus.org/wp-content/uploads/2017/10/226_Paper.pdf.

[5] Rodrigo Mira. Rl rnn composer. https://github.com/miraodasilva/RL_RNN_MusicComposer. Date of access: Wed, 29 Aug 2018.

[6] Elliot Waite. Generating long-term structure in songs and stories. https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn. Date of access: Wed, 29 Aug 2018.

[7] Florian Colombo and Wulfram Gerstner. Bachprop: Learning to compose music in multiple styles. *CoRR*, abs/1802.05162, 2018. URL http://arxiv.org/abs/1802.05162.

[8] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79 (8):2554–2558, 1982. ISSN 0027-8424. doi: 10.1073/pnas.79.8.2554. URL http://www.pnas.org/content/79/8/2554.

[9] Gino Brunner, Yuyi Wang, Roger Wattenhofer, and Jonas Wiesendanger. Jambot: Music theory aware chord based generation of polyphonic music with lstms. *CoRR*, abs/1711.07682, 2017. URL http://arxiv.org/abs/1711.07682.

[10] Chun-Chi J. Chen and Risto Miikkulainen. Creating melodies with evolving recurrent neural networks. In *Proceedings of the INNS-IEEE International Joint Conference on Neural Networks*, pages 2241–2246, Piscataway, NJ, 2001. IEEE. URL http://nn.cs.utexas.edu/?chen:ijcnn01.

[11] Nimesh Sinha. Understanding lstm and its quick implementation in keras for sentiment analysis. https://goo.gl/cwcEBG. Date of access: Wed, 29 Aug 2018.

[12] Douglas Eck and Jürgen Schmidhuber. Finding temporal structure in music: blues improvisation with LSTM recurrent networks. In *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing, NNSP 2002, Martigny, Valais, Switzerland, September 4-6, 2002.*, pages 747–756, 2002. doi: 10.1109/NNSP.2002.1030094. URL https://doi.org/10.1109/NNSP.2002.1030094.

[13] Douglass M. Green. *Intgral*, 3:227–234, 1989. ISSN 10736913. URL http://www.jstor.org/stable/40213921.

[14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL http://arxiv.org/abs/1409.0473.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6000–6010, 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.

[16] Qingtong Wu. A brief overview of attention mechanism. https://medium.com/syncedreview/a-brief-overview-of-attention-mechanism-13c578ba9129. Date of access: Wed, 29 Aug 2018.

[17] Hang Chu, Raquel Urtasun, and Sanja Fidler. Song from PI: A musically plausible network for pop music generation. *CoRR*, abs/1611.03477, 2016. URL http://arxiv.org/abs/1611.03477.

[18] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17286.

[19] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. Deepbach: a steerable model for bach chorales generation. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1362–1371, 2017. URL http://proceedings.mlr.press/v70/hadjeres17a.html.

[20] Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007. ISBN 9780387310732. URL http://www.worldcat.org/oclc/71008143.

[21] Kevin P. Murphy. *Machine learning - a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, 2012. ISBN 0262018020.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

[23] Sebastian Ruder. An overview of gradient descent optimization algorithms. http://ruder.io/optimizing-gradient-descent/index.html. Date of access: Wed, 29 Aug 2018.

[24] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *CoRR*, abs/1406.2661, 2014. URL http://arxiv.org/abs/1406.2661.

[25] Ian J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017. URL http://arxiv.org/abs/1701.00160.

[26] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

[27] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *CoRR*, abs/1611.00712, 2016. URL http://arxiv.org/abs/1611.00712.

[28] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *CoRR*, abs/1611.01144, 2016. URL http://arxiv.org/abs/1611.01144.

[29] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 2852–2858, 2017. URL http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14344.

[30] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2226–2234, 2016. URL http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.

[31] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, pages 41–48, 2009. doi: 10.1145/1553374.1553380. URL http://doi.acm.org/10.1145/1553374.1553380.

[32] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. URL http://arxiv.org/abs/1511.06434.

[33] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014. URL http://arxiv.org/abs/1411.1784.

[34] Christopher Harte, Mark Sandler, and Martin Gasser. Detecting harmonic change in musical audio. In *Proceedings of the 1st ACM Workshop on Audio and Music Computing Multimedia*, AMCMM '06, pages 21–26, New York, NY, USA, 2006. ACM. ISBN 1-59593-501-0. doi: 10.1145/1178723.1178727. URL http://doi.acm.org/10.1145/1178723.1178727.

[35] 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 *The 9th ISCA Speech Synthesis Workshop, Sunnyvale, CA, USA, 13-15 September 2016*, page 125, 2016. URL http://www.isca-speech.org/archive/SSW_2016/abstracts/ssw9_DS-4_van_den_Oord.html.

[36] Aäron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis C. Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. Parallel wavenet: Fast high-fidelity speech synthesis. *CoRR*, abs/1711.10433, 2017. URL http://arxiv.org/abs/1711.10433.

[37] Romain Sabathe, Eduardo Coutinho, and Björn W. Schuller. Deep recurrent music writer: Memory-enhanced variational autoencoder-based musical score composition and an objective measure. In *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*, pages 3467–3474, 2017. doi: 10.1109/IJCNN.2017.7966292. URL https://doi.org/10.1109/IJCNN.2017.7966292.

[38] Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1311–1320, 2017. URL http://proceedings.mlr.press/v70/graves17a.html.

[39] Bei Peng, James MacGlashan, Robert T. Loftin, Michael L. Littman, David L. Roberts, and Matthew E. Taylor. Curriculum design for machine learners in

sequential decision tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, pages 1682–1684, 2017. URL http://dl.acm.org/citation.cfm?id=3091403.

[40] Sébastien Forestier, Yoan Mollard, and Pierre-Yves Oudeyer. Intrinsically motivated goal exploration processes with automatic curriculum learning. *CoRR*, abs/1708.02190, 2017. URL http://arxiv.org/abs/1708.02190.

[41] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998. ISBN 0262193981. URL http://www.worldcat.org/oclc/37293240.

[42] Christopher J. C. H. Watkins and Peter Dayan. Technical note q-learning. *Machine Learning*, 8:279–292, 1992. doi: 10.1007/BF00992698. URL https://doi.org/10.1007/BF00992698.

[43] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

[44] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL http://arxiv.org/abs/1509.06461.

[45] C. Schmidt-Jones. *Understanding Basic Music Theory*. CreateSpace Independent Publishing Platform, 2015. ISBN 9781508534297. URL https://books.google.co.uk/books?id=T6orrgEACAAJ.

[46] William Thompson. Intervals and scales. pages 107–140, 12 2013.

[47] The midi association. https://www.midi.org/. Date of access: Wed, 29 Aug 2018.

[48] Tabplayer.online. http://tabplayer.online/. Date of access: Wed, 29 Aug 2018.

[49] Selenium - web browser automation. https://www.seleniumhq.org/. Date of access: Wed, 29 Aug 2018.

[50] Ole Martin Bjrndalen. Mido - midi objects for python. https://github.com/olemb/mido/. Date of access: Wed, 29 Aug 2018.

[51] Magenta. Melody rnn. https://github.com/tensorflow/magenta/tree/master/magenta/models/melody_rnn, . Date of access: Wed, 29 Aug 2018.

[52] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017. URL http://arxiv.org/abs/1712.04621.

[53] Meinard Müller and Frans Wiering, editors. *Proceedings of the 16th International Society for Music Information Retrieval Conference, ISMIR 2015, Málaga, Spain, October 26-30, 2015*, 2015. ISBN 978-84-606-8853-2.

[54] Magenta. Tuning rnns with rl. `https://github.com/tensorflow/magenta/tree/master/magenta/models/rl_tuner`, . Date of access: Wed, 29 Aug 2018.

[55] Magenta: Music and art generation with machine intelligence. `https://github.com/tensorflow/magenta`. Date of access: Wed, 29 Aug 2018.

[56] Martin A. Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Vlad Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing solving sparse reward tasks from scratch. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 4341–4350, 2018. URL `http://proceedings.mlr.press/v80/riedmiller18a.html`.

[57] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Deep inverse reinforcement learning. *CoRR*, abs/1507.04888, 2015. URL `http://arxiv.org/abs/1507.04888`.

[58] Aalf Gabrielsson and Erik Lindstrm. The role of structure in the musical expression of emotions. pages 367–400, 01 2010.

[59] Eduardo Coutinho and Nicola Dibben. Psychoacoustic cues to emotion in speech prosody and music. *Cognition & emotion*, 27(4):658–684, 2013.