



School of Mathematics, Computer Science & Engineering

MSc Data Science – 2016/17

Internship Project Report

**StuctLSTM:
Structure Augmented Long Short-Term Memory Networks
For Music Generation**

Author: Shakeel ur Rehman Raja

Supervisor: Dr. Tillman Weyde

(Submitted: 9th February, 2018)

DECLARATION

By submitting this work, I declare that this work is entirely my own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the coursework instructions and any other relevant programme and module documentation. In submitting this work, I acknowledge that I have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. I also acknowledge that this work will be subject to a variety of checks for academic misconduct. Marks are provisional and subject to change in response to moderation, assessment board decisions and any ongoing investigations of suspected academic misconduct.

Shakeel ur Rehman Raja

ACKNOWLEDGEMENT

I would like to express my deepest appreciation to all those friends and fellow data scientists who provided me help and support to complete this report. I would like to offer a special gratitude to my project supervisor, Dr Tillman Weyde, who provided original inspiration for this research idea and offered motivating suggestions and encouragement throughout the course of this project.

Furthermore, I would like to thank all of the participants who took part in the subjective evaluation stage of this experiment and helped me develop a better understanding of this domain through their responses and feedback.

I would also like to thank my father who has always been there for me and supported my passion for music and technology, and to my sons Umer and Ali, who bring great joy to my life. Last but not the least, I would like to thank my lovely wife Aliya for her love and support and giving me a reason to move on.

DEDICATION

I dedicate this work to my dear mother, who departed from us during the course of this research project, but left me with the gift of music which I shall always cherish and treasure. May God bless her soul.

.

ABSTRACT

Over the past few years, deep learning algorithms have shown state of the art performance in several challenging problems ranging from NLP, computer vision to robotics and driverless cars. The field of generative arts and machine creativity has also received special attention with growth in the popularity of generative deep learning. Deep generative algorithms have been used for music generation in a number of experiments demonstrating varying levels of success. Music modelling is considered a highly complex task since musical creativity and related cognitive processes are not fully understood, and thus very challenging to model. One of the key challenges in the area of music generation is to develop models that can learn structural and temporal aspects of a music corpus and generate structured musical sequences in a human-like fashion. This research attempts to address this issue with design and implementation of a deep recurrent neural network for generating structured musical sequences composed of unique melodies and harmonies. An RNN architecture with LSTM cells is used with Nottingham folk music dataset for training. The objective of this research is to study whether deep neural network models can learn to produce structured musical output by augmenting extra structural information to the input data. The experiment is performed in the framework of music language modelling, where the task of the model is to predict the next note and chord in the sequence. Feature augmented models are compared to a base model based on prediction loss. Objective evaluation of the model show that feature augmentation leads to an improved prediction. Generated sequences are subjectively evaluated using an on-line Turing style listening test, which also confirms the effectiveness and further research potential of proposed augmentation framework.

TABLE OF CONTENTS

Chapter1: Introduction	1
1.1 Background	1
1.2 Motivation	2
1.3 Purpose	2
1.4 Research Question	2
1.5 Research Objectives	3
1.6 Work Plan	3
1.7 Products	4
1.8 Beneficiaries	4
1.9 Changes in Objectives and Methodology	4
1.10 Report Structure	5
 Chapter 2: Critical Context	 6
2.1. Overview of Generation Techniques	6
2.1.1 Markov Models	6
2.1.2 Knowledge Based Systems	7
2.1.3 Evolutionary/Genetic Algorithms	7
2.1.4 Music Grammars	7
2.2. Neural Networks	8
2.2.1. The Neuron	8
2.2.2. Neural Network Architecture	9
2.3. Feed-Forward Neural Networks (FFNs)	10
2.3.1. Backpropagation Learning with FFNs	10
2.3.2 Music Generation with FFNs	11
2.3.3 Limitations of FFNs	11
2.4 Deep Learning for Music Generation	12
2.5 Recurrent Neural Networks (RNNs)	12
2.5.1 Backpropagation Through Time (BPTT)	13
2.5.2 Generative RNNs	15
2.5.3 Music Generation with RNNs	15
2.5.4 Limitations of Simple RNNs	15
2.6 Long-Short Term Memory Networks (LSTMs)	15
2.6.1 Music Generation with LSTMs	16
2.7 Challenges in Music Generation	17
2.8 Deep Learning Platforms	18
2.8.1 TensorFlow	18
2.8.2 PyTorch	19
 Chapter 3: Background	 20
3.1 Musical Background and Theory	20
3.1.1 Frequency / Pitch	20
3.1.2 Beats and Bars	20
3.1.3 Rhythm	21
3.1.4 Melody	21
3.1.5 Harmony	21
3.1.6 Long-term Structure	22
3.2 LANGUAGE MODELLING	22
3.2.1 Music Language Modelling	23

3.3	Music Representation	23
3.4	MIDI - Musical Instruments Digital Interface	24
3.4.1	Time Discretization	24
3.5	THE PIANO-ROLL REPRESENTATION	24
Chapter 4: Methods and Approaches		26
4.1	Supporting Tools and Technologies	26
4.2	Identifying a Code Base	26
4.2.1	Changes made to the Code Base	28
4.3	Exploratory Data Analysis	29
4.4	Data Parsing and Representation	30
4.4.1	Melody Encoding	31
4.4.2	Harmony Encoding	31
4.4.3	Input Encoding	32
4.5	Augmentation of Engineered Features	32
4.5.1	Augmenting Song Duration Counter	33
4.5.2	Counter Modifications	33
4.5.3	Augmenting Metrical Information	34
4.6	Mini-batching	36
4.7	StructLSTM (Structure Augmented LSTM)	37
4.8	Generation strategy	40
Chapter 5: Experiments and Results		41
5.1	Setting up the Base Model	41
5.1.1	Generation with Base Model	42
5.1.2	Zero-padding Input Data	43
5.2	Augmenting Sequence Duration Counter	44
5.2.1	Simple Integer Counter	44
5.2.2	Findings	46
5.2.3	Generation with Simple Integer Counter	46
5.2.4	Duration Control	47
5.2.5	Modified Counter with Normalization and Shift	47
5.2.6	Generation with Modified Counter	49
5.3	Augmenting Metrical Information	50
5.3.1	Augmenting Countdown and Metrical Information to Complete Dataset	50
5.3.2	Augmenting Metrical Information to 4/4 Sequences	52
5.3.3	Generation with Metrical Feature Augmentation	53
5.4	Model Comparison	54
Chapter 6: Subjective Evaluation and Results		56
6.1	Evaluation Framework	56
6.1.1	Identify Compositional Aims	56
6.1.2	Induce Human and Base Model compositions	57
6.1.3	Identify Candidate Compositions	57
6.1.4	Collect Listeners' Feedback	57
6.2	Online Listening Test	58
6.3	Subjective Evaluation of Generated sequences	60
6.3.1	Participant Groups	60
6.3.2	Average Scores	60

6.3.3	Evaluating Human Compositions	62
6.3.4	Evaluating Baseline Model Generation	63
6.3.5	Evaluating Counter Augmented Generation	63
6.3.6	Evaluating Counter and Metrical Feat. Augmented Generation	64
Chapter 7: Discussion		66
7.1	Fulfilment of Research Objectives	66
7.2	New Findings	67
7.3	Products Delivered	68
7.4	Research Question	69
Chapter 8: Evaluation, Reflection and Conclusion		70
8.1	Evaluation of Methods and Approaches	70
8.2	Future Work	72
8.3	Reflection	73
8.4	Conclusion	74
Glossary of Acronyms		75
References		76
Appendix A Initial Internship Proposal		80
Appendix B Pre-processing and Feature Augmentation		98
Appendix C Python Code for StructLSTM		100
Appendix D Generated Products		121
Appendix E Subjective Evaluation and Ethical Requirements		122

Chapter 1. INTRODUCTION

1.1 BACKGROUND

Recent advances in the field of *Artificial Intelligence* (AI), particularly in *Deep Neural Networks* have allowed researchers and practitioners from all major areas of study to develop state of the art models for *Pattern Recognition* and *Machine Learning* (ML) tasks as shown in (Schmidhuber, 2015) and (LeCun et al., 2015). Deep neural models are powered by the exponential rise in computational capabilities of computer hardware and enormous amounts of data being generated in almost all major research fields of study. This paradigm has also allowed experimentation with computational algorithms that can understand and model comparatively more complex data structures, as compared to traditional AI techniques. Examples of such data structures include spoken languages, visual objects and music as shown by (Deng and Yu, 2014), (Wang and Wang, 2014), (Karpathy, 2015), (Ngiam et al., 2011). In recent years, deep learning experiments including IBM's Watson¹ (High, 2012) and Google Brain Project² have clearly demonstrated that deep neural algorithms can compete and even outperform human beings in certain cognitive tasks.

Deep Learning Techniques have also demonstrated encouraging results when applied to the domain of generative arts. *Generative arts* are seen as an intersection of art, creativity and technology as shown by (Fernandez & Vico, 2013), (Karpathy, 2015), (Gregor et al., 2015) and (Denton, Chintala & Fergus, 2015). Generative models developed in these experiments are used for modelling languages, images and music in an attempt to understand and model human-like cognitive abilities towards these fields of arts.

In the context of generative arts, music generation is considered to be particularly challenging problem domain. Composing and appreciating music involves complex cognitive processes ranging from pitch detection, rhythm detection and structural inference, as well as a resulting subjective interpretation. These processes are not fully understood and are therefore very difficult to model in a computational framework, as discussed by (Briot & Pachet, 2017). A number of experiments on developing deep generative models to overcome these generation challenges have recently been performed with varying level of success (Boulanger-Lewandowski, Bengio & Vincent, 2012), (Coca et al, 2013), (Chung et al, 2014), (Jacques et al., 2016) and (Malik and Ek, 2017). These experiments have shown that learning music structure during training and demonstrating human-like understanding of musical structure towards composition is a highly challenging task and a rich area of research.

The context given above is used as a background and rationale for this research project. The project aims to experiment with a novel structure augmentation approach with a generative RNN and evaluate the results for effectiveness of the proposed methodology.

¹ IBM Watson (<http://www.ibm.com/watson>)

² Google Brain Project (<https://ai.google/brain-team>)

1.2 MOTIVATION

I have an academic and professional background in Computer Sciences and Information Technology, as well as a keen interest towards digital music composition and production using state of the art production tools. I carry hands on experience of production in modern studio environments in different parts of the world and in different music genres ranging from electronic music to Indian classical music. Over the years, I have developed a strong interest towards generative music stemming from my passion for technology and music. As a postgraduate student of Data Science at City, University of London, I had a chance to get introduced to Dr Tillman Weyde, who is a renowned music researcher, actively involved in research around music modelling and deep learning. Dr Weyde gave the original idea and inspiration for this research project and also supervised my research internship at Research Centre of Machine Learning at City, University of London for the duration of this project. I consider this a great learning opportunity as it allowed me to work with a domain that I am passionate about, under the supervision of a domain expert.

1.3 PURPOSE

The purpose of this research project is to design and implement a deep neural computational algorithm for solving a supervised music sequence learning problem in a language modelling framework. The experiment employs feature engineering and augmentation approach towards network training and attempts to control the structure, rhythm and duration of generated music. The proposed neural machine would thus facilitate knowledge representation in a connectionist learning paradigm, in an attempt to emulate human-like cognitive performance towards musical composition. The experiment uses objective and subjective evaluation to learn whether musical knowledge as extra features augmented into the network during network training and generation phase, can improve the structure of generated music.

The proposed approach is an innovative and novel research idea, and the results shown in the experiments could be of great interest to researchers and professionals involved in generative music with deep learning. It is also stated that this is an on-going research project for both the author and supervisor, and the results presented in this report are based upon current progress. The research has been conducted with an aim to publish this work at the end on internship period. Some details about planned future work have been provided in the conclusion of the report.

1.4 RESEARCH QUESTION

The primary research question that this project attempts to answer is stated as:

"Can we improve the performance of a Deep Recurrent Neural Network by augmenting engineered structural features to the training process in a Language Modelling framework, in order to stochastically generate musical sequences with improved structure and to achieve better control over the generative process"

1.5 RESEARCH OBJECTIVES

Based on the research question, the project work was split into a set of objectives to develop research framework, design experiments and evaluate the effectiveness of chosen approaches.

1. Research literature to identify AI and deep learning-based music generation approaches to develop a theoretical framework, and investigate deep learning tools and platforms to identify state of the art approaches.
2. Develop a baseline model to provide a comparison ground to measure the performance of proposed feature augmented models.
3. Create new datasets for experiments by applying necessary pre-processing to data for augmentation of engineered features to input encoding.
4. Set up a deep learning architecture and train it on different feature augmented datasets to measure and evaluate the model performance under different settings.
5. Stochastically generate new music from models trained on data sets developed earlier in an attempt to control the generation process with synthetic features.
6. Conduct a Turing styled human evaluation session to evaluate the improvement in the quality generated musical sequences, as compared to the baseline model.

1.6 WORK PLAN

Based on the objectives set in section 1.4, a work plan was developed to plan research activities around these objectives to be able to answer the original research question, as shown in Fig 1.1.

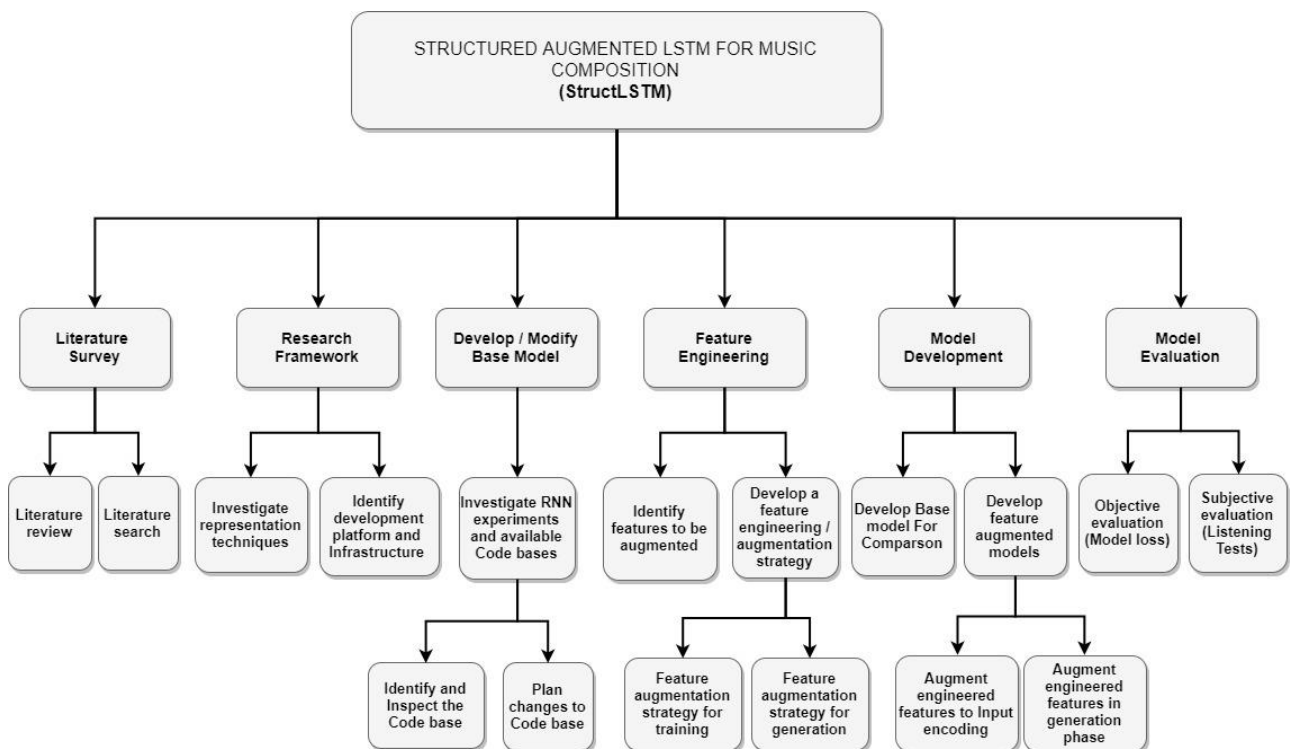


Fig 1.1 Work plan devised for achievement of research objectives set in section 1.4

1.7 PRODUCTS

Following products will be presented at the end of this research project:

- A pre-processing library for extracting melody, harmony and additional structural features from a MIDI corpus.
- A set of methods for engineering and augmentation of engineered structural features aimed at providing domain knowledge to the model.
- A feature engineering and augmentation framework that is applicable to both training and generation phases.
- A structure augmented, generative deep neural machine that would facilitate musical knowledge representation in a deep learning paradigm.
- Performance evaluation of the proposed system against base model using standard network loss calculation.
- A set of musical sequences generated as a result of different augmentation experiments.
- A subjective evaluation conducted through Turing style listening test to measure aesthetic improvement for generated music.

1.8 BENEFICIARIES

Potential beneficiaries of this work comprise following groups:

- Generative music application and systems developers and practitioners interested in finding novel ways to generate human-like music using computational approaches.
- Deep learning researchers exploring new ideas and methods towards improving the performance by adding domain knowledge to deep learning algorithms.
- Music informatics researchers keen to explore theories and practices of musical improvisations and performances through music information analyses and modelling.
- Musicians employing latest technology to aid their compositional work and live improvisatory performances.
- Tech companies employing AI techniques for music generation, in order to create custom music tracks to suit the needs of their clients.

1.9 CHANGES IN OBJECTIVES AND METHODOLOGY

The initial proposal set for this project was set under the title “*Neural-Symbolic Music Generation Using Probabilistic Musical Grammars*”. The objectives set within the initial proposal were aimed at developing a generative neural network with symbolic rule integration to govern production probabilities and also to enforce these rules during generation phase. Probabilistic context free grammars were chosen as means to define the rules using LTN (Logic Tensor Networks) library developed by (Serafini & Garcez, 2016). This library uses TensorFlow deep learning framework, to facilitate deep learning with logical reasoning. The risk register in the original proposal presented this as a risk of medium likelihood but with a very high impact. The original proposal and risk register has been appended to this report in **Appendix A** (Fig 3).

Upon initial literature review and investigation into rule implementation on temporal data (i.e. music) in a deep network, it was realised that the original proposal was highly over-ambitious to be completed within the given timeframe of internship. The LTN framework, in its current state, lacks the ability to deal with temporal data and only offers application in Feedforward networks for non-temporal data. Modifying this framework to suit the need of this experiment and applying the updated framework to the defined problem was hence identified as a very challenging task. After having discussions with Dr Artur Garcez (co-author of LTN paper) and Dr Tillman Weyde (project supervisor), it was decided to change the objectives of the initial proposal and to try alternative knowledge augmentation strategies.

This experiment, instead of using symbolic rules, uses engineered structural features during learning and generation phases. Although this approach is not as sophisticated and well-grounded as the one from the initial proposal, it is unique in a sense that this idea and related experimentation was not found during literature review of music generation with neural networks. Similar approaches, however, have been applied to Natural Language Processing tasks. This project hence provides an opportunity to evaluate the performance of feature augmented RNNs in the field of generative music.

1.10 REPORT STRUCTURE

This report is structured under following chapters.

- **Chapter 1** provides introduction, background, a research question and research objectives set for the research internship, with a workplan overview.
- **Chapter 2** contains a critical review of algorithmic music generation techniques with a focus on latest trends in deep learning with Recurrent Neural Networks and LSTMs.
- **Chapter 3** highlights some background knowledge on music theory and digital music representation formats which are later used for experimentations
- **Chapter 4** discusses the methods and approaches used towards developing a feature augmented neural architecture. It details data encoding, feature augmentation and generation techniques used towards building the StructLSTM model.
- **Chapter 5** details the experiments and an objective evaluation of the results achieved in terms of model performance and visual analysis of structure of generated music.
- **Chapter 6** contains information on developing a subjective evaluation framework, conducting an on-line listening test and evaluation of results obtained under the set framework. Chapter uses score statistics and visualization of results for evaluation.
- **Chapter 7** provides a discussion around achievement of research objectives, major findings and products delivered during experimentation.
- **Chapter 8** concludes the report with an overall evaluation of approaches, future work plan, author's reflection and a conclusion to the project.

Conclusion is followed by a list of referenced literature, and a set of appendices containing additional information on coding, experiments, products delivered and fulfilment of ethical requirements during the on-line survey.

Chapter 2. CRITICAL CONTEXT

Algorithmic music composition has been classically described as setting a sequence of rules for solving a problem of combining individual musical components into a complete composition. The term *Generative Music*, using computational algorithms was coined by Brian Eno's work, identifying it as a broad and rich category within the spectrum of generative arts (Eno, 1996). Since the inception of the idea of generative music, there have been a number of attempts to generate music using computational algorithms including *Markov Models*, *Genetic Algorithms*, *Natural Language Processing*, *Cellular Automata*, *Statistical Machine Learning* and *Neural Computing* as reviewed by (Papadopoulos & Wiggins, 1999), (Nierhaus, 2009) and (Fernandez & Vico, 2013). These methods have been employed by researchers and musicians with considerable success. However, as reported by these authors, there still exist many caveats such as in-coherent structure of generated music, lack of domain subjectivity, increased level of system complexity, absence of human-like improvisations, questionable quality of generated music and dependence of large training datasets and complex composition rules.

2.1 OVERVIEW OF MUSIC GENERATION TECHNIQUES

In order to fully appreciate the role of Deep Neural Networks, following section provides a brief overview of major intelligent music generation techniques that have been employed by researchers and practitioners in the past.

2.1.1 Markov Models

Mathematical models and stochastic processes, mainly *Markov Chains* have been used extensively for music generation since 1950s (Hiller and Isaacson, 1959). Iannis Xenakis used Markov models in his 1950s compositions using transition matrices to identify note production probabilities (Xenakis, 1992). David Cope combined Markov models with other computational techniques following the work of Hiller and Xenakis to create a semi-automated music system, as mentioned in "*Experiments in Music Intelligence*" (Cope and Mayer, 1996). The ability of Markov models to perform transitions in discrete time steps with a lower level of complexity allows them to be effectively utilized within real time music generation applications as shown in the Interactive jazz improvisations with *BoB System* (Thom, 2000). *Simulated Annealing* was used to generate rhythm and melody in order to apply musical constraints on the generative process by Davismoon and Eccles, can be seen as natural evolution of such models (Davismoon & Eccles, 2010).

These models, in general, can produce music highly similar to that in the original dataset. In order to exhibit the ability to deal with higher abstractions found in human music, Markov models need large amounts of data for probabilistic computations to become generalizable.

2.1.2 Knowledge Based Systems

Knowledge based systems, or KBSs, are primarily symbolic in nature, having explicit rules and well defined structural constraints to control the output of the model in the form of a knowledge base. An *Inference engine* enables a knowledge-based system to imitate the actual processes that a composer engages in when applying these musical rules. The main advantage of such systems is their ability to explain choices and actions taken to generate a certain output. The rule-based harmonization system, *CHORAL* (Ebcioglu, 1988) and *Intention-based Music System* (Zimmerman, 1998) can be seen as noteworthy examples of KBSs for generating compositions. The main limitations for such systems are an increased level of complication towards knowledge elicitation and a high dependency on the skills of experts for encoding these rules into computational models.

2.1.3 Evolutionary/Genetic Algorithms

Evolutionary algorithms, inspired by evolutionary biology, have proven to be an effective alternative to some of the techniques mentioned above due to their ability to work with large search spaces and provide multiple unique solutions which is a desirable function while dealing with music generation. These algorithms are inspired by the idea of biological evolution and are known to solve difficult problems in multiple domains using evolutionary optimization algorithms. These algorithms have been applied to a number of harmonisation and accompaniment experiments. *Genetic programming*, which is a commonly used evolutionary technique, has proved to generate programs that produce rhythmic melodies as output when given a melody as an input. One of the early examples in this regard is by Horner and Goldberg where authors implemented evolutionary methods for musical *thematic bridging*, allowing thematic musical content that can be defined to a genetic algorithm (Horner & Goldberg, 1991). Also, professional score writing with *Iamus* (Diaz-Jerez, 2011) can be seen as a good candidate example.

These methods are capable of producing aesthetically pleasing melodies but lack subjective elements and exhibited lower efficiency due to “*Fitness Bottleneck*” problem as shown by Biles with *GenJam*, a genetic algorithm for creating jazz solos (Biles, 1994).

2.1.4 Musical Grammars

A Musical Grammar is a set of rules that governs musical structure by expanding high level symbols into detailed sequence of melodies (Baroni, Maguire & Drabkin, 1983). The approach towards musical grammars takes its inspiration from grammatical models developed by cognitive linguists like Ronald Langacker (Langacker, 1957). Grammar based symbolic music generation emerged from Generative Theory of Tonal Music (GTTM) developed in 1980s (Lerdahl & Jackendoff, 1985). GTTM allowed *Context Free Grammars* (CFGs) for probabilistic modelling of structural elements like repetition, transformation and note-variation. Grammars are also useful towards enforcing music-theoretic ideas such as scale, harmony and tonality etc. into a computational model as shown in (Abdallah & Gold, 2014) and (Groves, 2016). Grammar derivation can recursively reduce or produce a melody by eliminating or adding melodic elaboration notes. *Probabilistic Context Free Grammars* (PCFGs) are used by Baroni (Baroni &

Callegari., 1984) for generating music in the style of Lutheran chorales. The logical probabilistic framework, *PRISM* (Abdallah and Gold, 2014) for automatic music analysis and reduction, *GTTM Analyser* (Hamanaka, Hirata & Tojo, 2015) for unsupervised PCFG based music generation, and automatic melodic reduction and generation (Groves, 2016) with supervised learning of PCFGs can be presented as key examples in this context. Another type of musical grammar called *Probabilistic Temporal Graph Grammars* (PTGGs) has been implemented in a generative system by Donya Quick, called *Kulitta* (Quick, 2014). PTGGs incorporate both PCFGs and as well as support for high level music structures. This grammar incorporates both metrical structure and pattern repetition by using “*Chord Spaces*” and genre-specific improvisations towards harmonic and melodic music sequences.

Some limitations of musical grammars have been highlighted as being computationally expensive, over-all questionable quality of an infinite number of generated melodies and expertise needed towards defining music theoretic concepts as grammar rules as evident from Hamanaka’s GTTM analysis experiments (Hamanaka, Hirata & Tojo, 2015).

In recent years, most of the experiments performed in the area of generative music have been employing neural networks, particularly deep networks for music generation. Following section focus on this approach while highlighting the key concepts that will be used for developing a theoretical and development framework.

2.2 NEURAL NETWORKS

A *Neural Network* is a “*massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use*” (Haykin, 2004).

Neural computation gets its inspiration from the biological functions of human or animal brain and the central nervous system (Aleksander & Morton, 1990). *Artificial Neural Networks*, or ANNs, built as a large network of signal processing components try to mimic the connections of neurons within brain, can perform a number of tasks including pattern recognition, anomaly detection, signal processing, predictions and forecasting.

Since 1980s, ANNs have gained special attention by musicians and music informatics researchers because they can be trained on complex structured musical patterns present in a given musical dataset and generate melodic content with improved structure, better control over model definition and generation processes, realistic improvisation and higher flexibility in design as shown by (Fernandez & Vico, 2013) and (Briot & Pachet, 2017). Following text attempts to summarize the basic principles of Neural Networks. The description is kept to such a level that ensures that all the terminology used within the experiments gets defined.

2.2.1 The Neuron

In a Neural network, the *Neuron* is the fundamental unit of computation. The idea of Neuron originated with McCulloch and Pitts Neuron (McCulloch & Pitts, 1943). A number of these neurons are connected by links known as weights that perform the function of neural synapses found in the biological brain and are responsible for neural activations. The representation of a

biological neuron as a computational unit, as described by Haykin is given in Fig. 2.1 (Haykin, 2004).

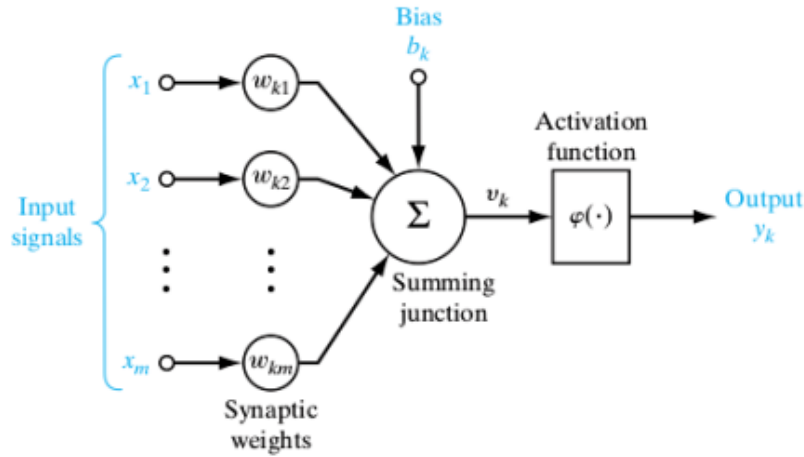


Fig. 2.1 A neuron, the basic unit of computation within a neural network. A Neuron is connected to its environment through links or weights which perform synaptic functions, found in biological nervous systems (Haykin, 2004).

A neuron k is initialized with random weights. For every example, all the inputs to the neuron are multiplied with respective weights. At the *summing junction* shown as Σ , the net input v_k is calculated by performing a sum of all inputs x_i . An extra input, known as the *bias* is used to control the behaviour of the neuron and is connected to neuron with a weight similar to inputs. The function of summing junction is shown in Equation 2.1.

$$v_k = \sum_{i=1}^m w_{ki} x_i + b_k \quad \text{Equation 2.1}$$

To calculate the output, neuron performs some non-linear computation function $\varphi(\cdot)$ on net input as shown in the figure. This is called the *activation function* and impacts the output y_k of neuron with given inputs. The final output is calculated as shown in Equation 2.2.

$$y_k = \varphi(v_k) \quad \text{Equation 2.2}$$

2.2.2 Neural Networks Architecture

Neurons can be inter-connected in a number of ways to develop a *Neural Network*, which be identified by a network architecture. A *network architecture* is defined as a relationship between neurons by means of their connections and is made up of a *neural framework* and *interconnection structure* (Fiesler, 1996). Neural framework is developed by setting a number of neural layers and defining number of neurons for each neural layer. A number of Neural layers can be connected to form a *Multi-layer network*. Depending on the position of a layer within the network, it can be identified as *input*, *output* or *hidden* layer. Hidden layers are used to learn the non-linear patterns within the data. Interconnection structure is defined by the way in which different neurons are linked within the network i.e. *feedforward*, *feedback* or *recurrent/self*-connections. Different types of network architectures have been developed and tested for solving a number of problems using pattern recognition and machine learning techniques. The network architecture plays an important role towards its function and

performance allows combined effect of individual neurons' activations to govern the overall behaviour of an ANN. A widely used, yet simple ANN architecture is a feed-forward network comprised only of feedforward connections.

2.3 FEED-FORWARD NETWORKS (FFNs)

A *Feed-Forward Network* (FFN) is a simple neural network architecture where connections between different neurons are feedforward and do not form a cycle. A *multi-layered feedforward network* can use any number of hidden layers between input and output layer to improve the decision boundary. FFNs having a single hidden layer with neurons using a nonlinear activation function can act as *universal function approximator* i.e. it is able to approximate any function as shown in (Hornik, Stinchcombe & White, 1989). Using hidden layers, a network can learn “*high order statistics and a global perspective*” of the input data. (Haykin, 2004). Example of a simple with FFN with a single hidden layer is shown in Fig.

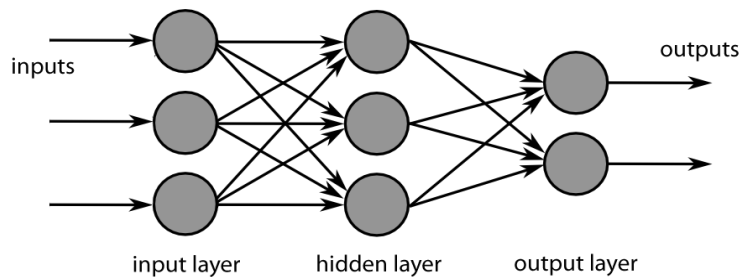


Fig 2.2 A feedforward network with one hidden layer can act as universal approximator.

2.3.1 Back-Propagation Learning with FFNs

A neural network learns to approximate a function in a supervised learning paradigm, where the synaptic weights between neurons are iteratively modified in a way that output of the network matches the ground truth (Haykin 2004). *Backpropagation* learning method (LeCun, 2015) is common technique used within neural architectures allowing network to learn from loss in prediction for each example.

A neural network starts processing with random weights and bias values for all neurons. The network is trained with a training dataset and corresponding target values. A *Forward Pass* measures the difference of network's output and target values in order to calculate *Network Loss*. Squared error is a commonly used loss function for FFNs. Backpropagation offers a differentiation method in FFNs to calculate the contribution of each neuron towards the total loss after the forward pass. In neural computing, backpropagation is used with *Gradient Descent* (GD) algorithm to adjust the synaptic weight of neurons by calculating the gradient of the loss function. This step is called the *backward pass* as the loss is calculated at the network output and propagated back through the network layers.

The mathematical details of the backpropagation and gradient descent are outside the scope of this research project. The details have been summarized in Fig 2.3 highlighting that after a

forward pass, error for each neuron is calculated based on the prediction error (or network loss) shown in (a), and propagated back to the network allowing weight updates, given in (b).

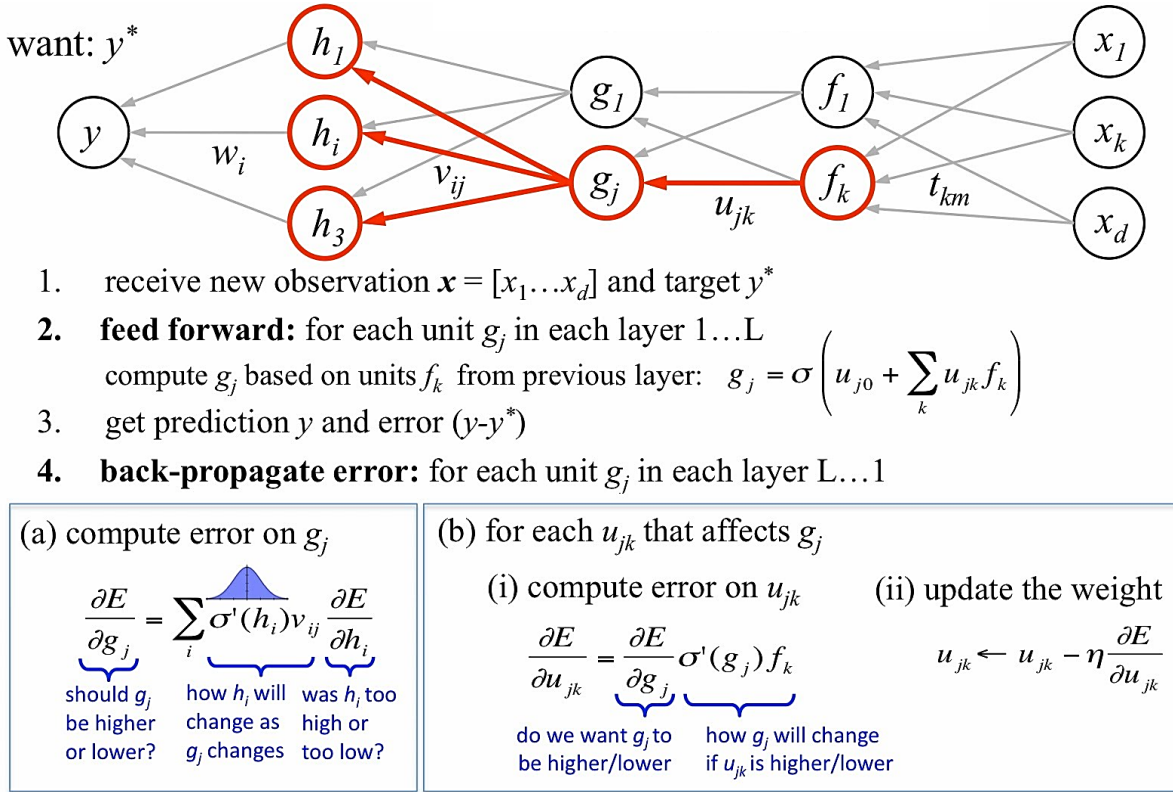


Fig. 2.3 Backpropagation Algorithm for training a feed forward network (Lavrenko, 2015).

2.3.2 Music Generation with FFNs

Using a feedforward neural architecture, Hadjeres and Briot's *MiniBach Chorale Generation System* follows a single step feedforward strategy to generate accompanying melodies in the style learnt from a corpus containing J. S. Bach's polyphonic music. This system uses a single hidden layer with Relu activation function and calculates the loss using a soft-max loss function as highlighted in *Deep Learning Techniques for Music Generation-A Survey* (Briot, Hadjeres & Pachet, 2017).

2.3.4 Limitations of FFNs

According to authors of above experiment, the music generated by this system does not sound convincing due to the fact that simple FNNs cannot take into consideration, the temporal association between musical notes. The lack of ability to deal with time-series data makes feed forward networks unsuitable for music generation experiments on their own. An improved version of this system called *DeepBach* was later developed that employed deep learning techniques on same data and producing much more convincing results (Hadjeres & Pachet, 2016). In other experiments, FFNs have been used in conjunction with evolutionary algorithms, Rule-based systems and Markov chains as explained by (Fernandez and Vico, 2013). Such

complicated architectures improve upon the performance of simple FFNs, however due to their complexity and emergence of deep learning, these are not heavily researched upon at present.

2.4 DEEP LEARNING FOR MUSIC GENERATION

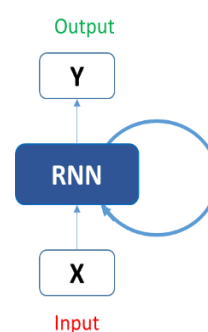
Neural computational architectures containing more than one hidden layers of neurons between the input and out layers are referred to as *Deep Neural Networks* (LeCun et al., 2015). As mentioned earlier, hidden layers in neural networks usually perform a series of successive non-linear transformations on the input data as it flows towards the output layer. Early experiments with ANNs attempted to compose monophonic melodies based on probabilistic modelling. In recent years, *Deep Neural Networks*, including *Deep Feed-Forward Networks* (DFFNs), *Restricted Boltzmann Machines* (RBMs) and especially *Recurrent Neural Networks* (RNNs) have gained particular popularity in the field of music generation as they allow representation of complex musical structures including polyphonic music and have the ability to generate sequences with improved temporal and structural dimensions.

Some other noteworthy mentions in this context of deep learning for music generation are polyphonic music generation with temporal dependencies using RNN-RBM hybrid architecture (Boulanger-Lewandowski et al., 2012) and Convolutional Generative Adversarial Networks for symbolic learning and generation (Yang, Chou and Yang, 2017). Google Magenta¹, a project of Google brain team, is the current state of the art for developing machine learning models for generative arts including images, language and music. One noteworthy project under this platform is deep *Re-Enforcement Learning* (RL) for music generation with RL-TUNER (Jacques et al., 2016), to improve the structure and quality of sequences generated by an RNN. The following section of this report will focus on suitability of recurrent neural networks for music generation.

2.5 RECURRENT NEURAL NETWORKS (RNNs)

Recurrent Neural Networks, commonly known as RNNs, are a specific type of neural algorithms where the connections between neurons form directed cycles which helps them incorporate temporal aspects of data (Haykin, 2004). RNNs have the ability to capture temporal dependencies between input examples. In an RNN, the output of hidden neurons is fed back as its own input allowing it retain a *Memory* as shown in Fig 2.4.

Fig 2.4 A simple RNN where the output from previous time step is fed back as input, thus allowing the network to retain memory of the past events



¹ Google Magenta (<http://magenta.tensorflow.org>)

Recurrent connections allow the state of a neuron at time $t - 1$ to be used as a hidden state input at time t , along with new input example in order to calculate the next output for time t , and a hidden state for time $t + 1$. This mechanism enables RNNs to perform temporal processing and learn changes in sequences over time or space making them a suitable candidate for processing time-series data. RNNs are hence considered state of the art for problems in sequence learning, sequence reproduction, temporal relations learning and time-series prediction. Theoretically, as the network grows deeper i.e. more layers are added to the network, each layer learns high level abstract representations of data and temporal relationships between input sequences.

The computational power of RNNs has been investigated by (Siegelmann & Sontag, 1995), which shows that for any computable function, an RNN model exists that can compute it. The paper also states that some RNN architectures can have the ability to simulate any algorithm makes RNNs *Turing complete*. However, the theory does not provide a clear way of developing an RNN that can achieve this. RNNs, much like MLPs and other machine learning models, use training data to learn during learning phase to update its parameters.

2.5.1 Back-Propagation Through Time (BPTT)

For loss calculation, an RNN is *Un-folded* along the time axis into a full network for the complete sequence seen at the input as described in Fig. 2.5. An unfolded RNN resembles a FFN provided a finite set of inputs is used. The learning algorithm it uses is a modification of backpropagation called *Backpropagation Through Time* or BPTT.

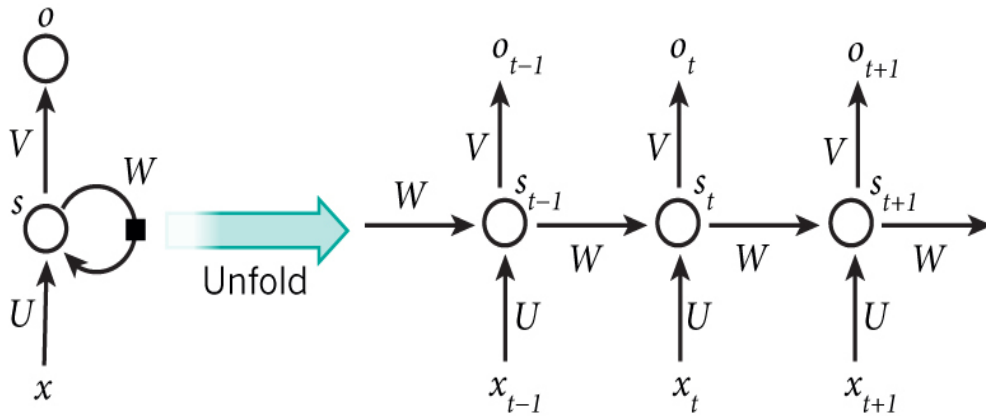


Fig. 2.5 A Unfolded RNN (source: nature)

Parameters that are modified during the learning process of an RNN are weights *from input to hidden layer* U , weights from *hidden to hidden layer* W and weights from *hidden to output layer* V . A *state update* step updates current state of the network the by taking product of input x_t and weight U and adding to the product of previous state s_{t-1} with weight W shown in Equation 2.3. An activation function d is applied at every step of status update. Output is based on previously updated state and hidden to output weight V as given in Equation 2.4.

$$s_t = d(Ux_t + Wx_{t-1}) \quad \text{Equation 2.3}$$

$$o_t = Vs_t \quad \text{Equation 2.4}$$

Calculating these two equations completes the *forward-propagation step*.

After the completion of the forward pass, the network loss against ground truth is calculated and backpropagation step or *backward pass* is performed. A typical loss function used for loss calculation in RNNs based classification and sequence learning problems is *Cross Entropy loss* or *Log Loss*. It is used to measure the loss of a model whose output is a probability value between 0 and 1. The more an output diverges from its actual class, the higher cross entropy becomes. For an output o_t with ground truth \hat{o}_t , cross entropy loss E is calculated as given in Equation 2.5.

$$E_t(o_t, \hat{o}_t) = - \sum o_t \log \hat{o}_t \quad \text{Equation 2.5}$$

A variation of gradient descent called *Batch Gradient Descent* (BGD), performs error calculation for each example in the training data and updates the model only after a set of all training examples, known as an *Epoch*, have been shown to the model. This increases computational efficiency, but may become slow for large datasets. *Mini-batch Gradient Descent* splits the data into smaller batches of defined length called *mini-batches* which are used for loss calculation and weight updates with BPTT, resulting as efficient training.

BPTT learning algorithm unfolds RNNs to a finite number of time steps to calculate the loss of the output predicted against the ground truth. For calculating the error relative to weight, BPTT calculates the sum of the gradients obtained for weights w_k in equivalent layers.

$$\frac{\partial E}{\partial w_k} = \sum_{t=1}^T \delta_k^t o_{k-1}^t \quad \text{Equation 2.6}$$

The partial derivatives in Equation 2.6 used for weight updates are calculated by multiple instances of input-to-hidden and hidden-to-hidden weights, thus depending upon the input and hidden neural states and preceding time steps. This setup allows to back propagate the error through the network as well as through time.

A *Deep Recurrent Neural Network* is formed by stacking up many recurrent layers. Each layer in such a network performs non-linear transformations on the input sequence to generate output sequence which is then used as input to a deeper layer at consequent time step.

2.5.2 Generative RNNs

RNNs could be *discriminative* or *generative* based on how input and outputs are related to each other (LeCun, 2015), (Karpathy, 2015). Generative RNNs can understand and internalize the relationship between training examples, in order to generate similar examples.

The literature review identified the most popular method to qualitatively evaluate the performance of a generative RNN is to generate musical sequences as network's output by *Conditioning* or *Priming* it for a small number of time steps with arbitrary information. The output of the conditioned model is sampled to generate new sequence which is fed back into the network, for a number of predefined repetitions. *Sampling* is the process of generating a sample from a probabilistic model with a defined probability distribution (Briot, Hadjeres & Pachet, 2017).

2.5.3 Music Generation with RNNs

The first major experiment for music generation using RNNs was performed by Mozer in 1989 with *CONCERT* system (Mozer, 1994). Mozer trained this system on soprano voices of Bach chorales, folk melodies and waltz harmonies to compose new musical pieces. Using simple RNNs, note pitch, note duration and chord structure were augmented into the model using a psychologically grounded representation. The *CONCERT* system is able to reproduce short term coherence but lacks a global coherence and according to Mozer, it is "*occasionally pleasant*". The lack of a long-term structure limits the potential use of this system.

2.5.4 Limitations of Simple RNNs

One of the key reasons behind lack of *global structure* or a *long-term structure* is that the network cannot understanding long term dependencies between examples of input data. A simple RNN suffers from the problem of *exploding gradients* as mentioned in (Pascanu, Mikolov & Bengio, 2012) while computing backpropagation through time. Simple RNNs are based upon *Tanh* function to compute the changes in weights at each time step during learning process. The limited representation of the coherent musical structure, as reported by Mozer, is due to the due to the fact that *Tanh* function allows the network to only focus on short term relationships while avoiding the exploding gradient problem. Sophisticated memory cells like *Long Short-Term Memory Networks* (LSTMs) *Gated Recurrent Units* (GRUs) are commonly used instead of simple activation functions to deal with this issue. For experimentation, the report will present a context on LSTMs in the following sections.

2.6 LONG-SHORT TERM MEMORY NETWORKS (LSTMs)

LSTM cells add complexity to a simple RNN cell by using special *Gates* to allow the cell to remember meaning information for longer periods of time (Hochreiter & Schmidhuber, 1997). Five components are typically used by LSTMs namely *Input gate* *i*, *Output gate* *o*, *Forget gate* *f*, *Cell memory* *c* and a *Hidden State* *h* at a given time step *t* as shown in Fig 2.6.

The LSTM based solutions are found to be very effective towards learning temporal relationships, while avoiding the exploding gradient problem in RNNs (Karpathy, 2015). The motivation behind an LSTM cell is allowing the RNN to store and preserve information in special memory cells grouped as a block. The usual flow of data can not affect the state of these cells until required.

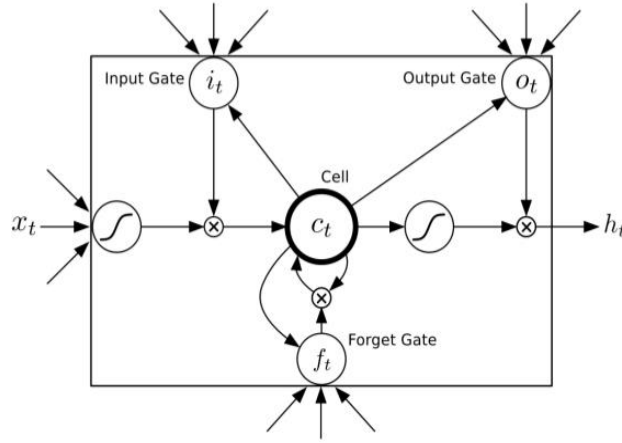


Fig. 2.6 An LSTM block with input, output and forget gates allowing the cell to store meaningful memories for longer periods of time while avoiding the exploding gradient problem as with simple RNNs.

The decisions to *Read*, *Write* and *Forget* are executed by opening and closing of input and output gates that learn to perform these functions during the learning phase. Gates are governed by weights and using back propagation techniques, the LSTM block architecture learns the memory management as a function of input data while minimizing its loss. The gating approach also solves the vanishing gradient problem by allowing a cell to withhold information for longer periods of time which helps the purpose of BPTT.

An investigation into mathematical details of an LSTM memory management technique is outside the scope of this research project. Following passage highlights major experiments performed with LSTM based RNN architectures. For some experiments particularly LSTMs, basic architectural details are highlighted for later reference and to develop a context for further experimentation.

2.6.1 Music Generation with LSTMs

In 2002, Doug Eck and Jürgen Schmidhuber updated Mozer's approach by switching from a simple sigmoid activation based RNN to more sophisticated LSTM architecture as described in (Eck & Schmidhuber, 2002). Authors use this architecture to generate blues melodies with a focus on maintaining long term structure of the output. The experiment is performed with a piano-roll representation of data for melodies and harmonies. Harmony chords are presented in the network as combination of notes rather than classes. Eck uses *one-hot encoding* with *time quantization* set to 8th note as half on minimal note duration found in the corpus. The architecture of this network has an input and output layers of same size with a single hidden layer. The generative part of the experiment is performed by *priming* the network with a seed and iteratively predicting the next note. In a second experiment with an extended architecture,

both harmony and melody are learnt at the same time. According to Eck, “*LSTM is able to play the blues with good timing and proper structure as long as one is willing to listen.*” (Eck and Schmidhuber, 2002)

Above experiment, considered to be the first major experiment in music generation using LSTM based RNNs, still has its limitations including overly simplified data, limited output classes and deterministic output. Google Magenta Project for generative arts has used Eck’s LSTM based approach for melody generation, polyphonic music generation and drums tracks generation with further modifications and bigger datasets.

The true potential of RNNs towards generative arts, particularly in language processing was highlighted by Andrej Karpathy’s article titled “*The Unreasonable Effectiveness of Recurrent Neural Networks*” (Karpathy, 2015). Karpathy highlighted that a simple recurrent neural architecture called *char-rnn* was able to recreate the “look and feel” of any input text corpus. Karpathy’s approach towards text generation was applied to music by Bob Sturm to use *char-rnn* with 23,000 pieces of Irish folk music in *ABC format*, a textual representation of music (Sturm, Santos & Korshunova, 2015). Daniel Johnson modified the RNN architecture and created *biaxial RNNs* that used a *note axis* and a *time axis* to learn the temporal dependencies between musical notes (Johnson, 2015). Christian Walder used LSTM networks with custom encoding technique instead of a standard piano roll representation. This experiment managed to capture some global network but due to application of heavy constraints on the generation process, the output was highly predictable (Walder, 2016). Neural style transfer was reported by (Malik & Eck, 2017) for learning music dynamics from data corpus and transferring the learning to other datasets.

While LSTMs can perform much better than simple RNNs and Markov models towards learning and maintaining long term structure of sequences, they still lack the ability of generate a complete composition with a global structure. Further human intervention and fine tuning of such models is a common practice towards developing a composition to make it sound human-like. Some recent start-up companies including Amper¹, Aiva² (Artificial Intelligence Virtual Artist), Melodrive³ and Jukedeck⁴ use such fine-tuning to generate custom music based on predefined set of themes to cater for their clients’ needs. Flow Machine, a state of the art AI based music composition system, developed by François Pachet (Briot & Pachet, 2017) uses proprietary deep learning algorithms to explore new ways of creating music.

2.7 CHALLENGES IN MUSIC GENERATION

Based on the literature review, the primary challenge identified towards music generation is the difficulty for network architectures to understand temporal and structural dimensions of music during network training and ensuring a structured output in the generation phase. The

¹ Amper Music (<http://www.ampermusic.com>)

² Aiva Technologies (<http://www.aiva.ai>)

³ Melodrive (<http://www.melodrive.com>)

⁴ Jukedeck (<http://www.jukedeck.com>)

lack of human-like performance due to lack of necessary structural elements and resulting subjective interpretation of generated music makes music modelling a difficult problem domain.

Simple Feedforward neural networks lack the ability to process different input sequences along with their temporal associations and thus would consider these sequences to be independent of each other. This provides us ground to consider Recurrent Neural Networks for the proposed experimentation. A simple RNN, however, would suffer from the problem of *exploding gradients* as mentioned in (Pascanu, Mikolov & Bengio, 2012) while computing backpropagation through time. Sophisticated memory cells like *Long Short-Term Memory Networks* (LSTMs) (Hochreiter & Schmidhuber, 1997) and *Gated Recurrent Units* (GRUs) (Chung et al., 2014) have proven to be very effective towards learning temporal relationships over smaller periods of time. However, ensuring the presence of global or the long-term structure in the generated music still requires fine tuning and post-processing of generated sequences. Different encoding techniques and architecture are currently being employed by researchers and practitioners to enhance the learning process. Augmentation of domain specific knowledge into the training process is another approach which is heavily being applied to deep learning in a number of application domains as mentioned by (LeCun, 2015). Feature learning and engineering approaches as mentioned by (Bengio, Courville and Vincent, 2012) are also gaining momentum with deep learning architectures.

Due to the state of the art nature of LSTM based generative architectures, it has become the de-facto standard in language modelling and sequence learning problems, and therefore shows a great potential for experimentation like feature engineering and developing novel generation techniques.

2.8 DEEP LEARNING PLATFORMS

At present, a number of different deep learning programming environments are available that are being used for state of the art application development in pattern recognition and machine learning problems. Deep learning is currently considered state of the art in machine learning and most deep learning platforms are open source software. Two such popular frameworks are highlighted in this report with their merits and limitations. The main criteria for selecting a framework was identified as availability of learning resources as these environments and related programming practices may get highly complex.

2.8.1 TENSORFLOW¹

TensorFlow is developed by researchers at Google Brain team for deep neural research and application development. The framework is highly generalized so it can be applied to a number of problems domain. TensorFlow is considered the most documented deep learning framework, which plays an important role towards its adoption as deep neural networks can become very complicated to understand and program while maintaining efficiency in

¹ TensorFlow (<http://www.tensorflow.org>)

approaches. TensorFlow uses Python, which is a widely adopted programming language and thus reduces the barrier to entry for programmers who have interest in deep learning. TensorFlow provides support for a variety of deep neural architectures including RNNs with GRUs and LSTMs, CNNs, RBMs and Deep Autoencoders etc. This framework also offers support for graphical representation of complicated networks through TensorBoard which is a network visualizer. TensorFlow offers GPU based deep learning for speeding up the network training time.

TensorFlow offers a vast set of experiments and examples conducted within generative arts domain. A number of music generation experiments using symbolic as well as raw audio music representations have been performed with TensorFlow.

2.8.2 PyTorch ¹

PyTorch offers a python based deep learning platform capable of performing complex tensor computations for deep neural networks. PyTorch, originating from Lua based Torch ² development framework, uses Dynamic Graph Computation as compared to Static Graph computation offered by TensorFlow, which can potentially allow design and implementation of deep learning models with relatively less complex programming routines, and with a higher transparency and flexibility. PyTorch was officially launched in 2017 has gained enormous popularity for NLP researchers.

Due to PyTorch being a very recent platform, the amount of help and support currently available for music generation is still almost non-existent. It does, however, show a lot of potential towards development and experimentation of flexible generative neural networks.

Other platforms, including Theano³, Caffe⁴, Keras⁵ and Microsoft Cognitive Toolkit⁶ are also being widely used by deep learning community. TensorFlow, however, remains a go-to platform for newcomers in the field of deep learning due to strong backing by Google and a highly mature set of available resources and hence considered suitable for this experiment. PyTorch, is currently being promoted by Facebook and shows great potential for future growth.

¹ PyTorch (<http://pytorch.org>)

² Torch, Scientific Computing for Lua|IT (<http://torch.ch>)

³ Theano (<http://deeplearning.net/software/theano>)

⁴ Caffe, deep learning framework (<http://caffe.berkeleyvision.org>)

⁵ Keras (<https://keras.io>)

⁶ Microsoft Cognitive toolkit (<https://www.microsoft.com/en-us/cognitive-toolkit>)

Chapter 3. BACKGROUND

3.1 MUSICAL BACKGROUND AND THEORY

The following section will attempt to provide an overview of musical concepts and principles which will be later implemented within the proposed deep learning architecture. The description is kept to a bare minimum to provide an introduction to some basic terminology. A detailed description of music theory and music formats is beyond the scope of this project.

3.1.1 Frequency / Pitch:

A musical composition can be presented and understood as a sequence of musical notes where each note can be defined using a number of attributes in the context of the composition it belongs to. All musical notes are labelled using 8 alphabets ranging from A to G, as their pitch classes, separated by classically identified frequencies as shown in Fig 3.1. A pitch interval is defined by the difference in pitches of two musical notes.

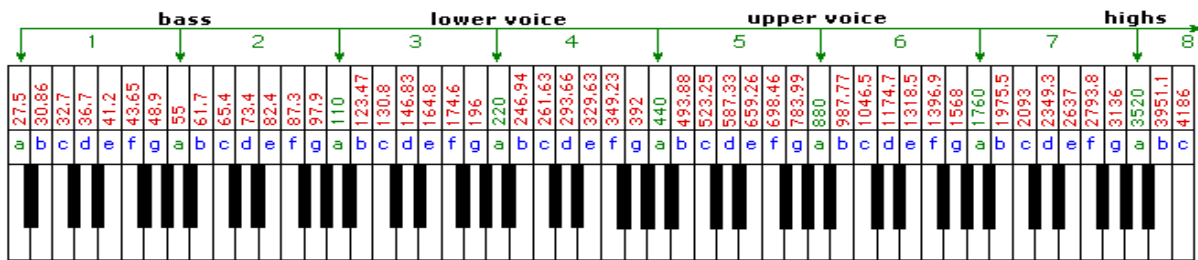


Fig 3.1 pitch classes, ranging from A to G are identified by their frequencies.

3.1.2 Beats and Bars

Short-term musical structure is defined by the note combinations in adjacent bars. A *beat* is a basic unit of time for a musical piece. The *tempo* defines the pace of the piece, by specifying how fast the beat is. Modern music also uses metronome markings, by specifying the number of beats in a minute, in *bpm* (beats per minute). The pace of a beat generally remains constant throughout the composition. A musical *bar* can be defined with a *time signature*, which in sheet music, is represented at the beginning of the piece, after the clef and key signature, as shown in Fig. 3.2. A time signature is shown using two numbers where the top number defines the number of *beat divisions* present in a bar and the bottom number defines the type of note (in terms of duration) to get one beat. A time signature of 4/4 explains that there will be four beats (specified by the top number), with each being a quarter note long (specified by the bottom number).



Fig. 3.2 An example of a 4/4 bar. A 4/4 bar consists of four crotchets, each being a quarter beat long.

3.1.3 Rhythm

Rhythm refers to the duration and accent given to individual music notes. It determines the duration of every note in a musical piece by using different symbols for different durations, as shown in Fig. 3.3. It also allows the composer to emphasize certain notes over others, to define the scale in which the music is being played. Rhythm is arguably the essential component of short term structure in a composition, as it can easily split the music into sections, while working alongside melody, is capable of delivering a plethora of emotions.















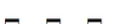
Name	Note	Rest	Beats	$1\frac{4}{4}$ measure
Whole			4	
Half			2	
Quarter			1	
Eighth			$\frac{1}{2}$	
Sixteenth			$\frac{1}{4}$	

Fig 3.3 Representation of note durations, and how they relate to a 4/4 bar

3.1.4 Melody

A *melody* (commonly referred to as the tune of a song) is a sequence of notes in a single voice which can be created using human vocals or other musical instruments around a defined rhythm. Melody defines structure of the music, which is considered to be the most prominent aspect of a composition, and is the key component in shaping the composition. Melody usually serves as the focus of the composition and a mode of communication for the composer to communicate emotions to the audience by controlling and modifying the melodic contour (melodic motion) of the melody as rising, falling, flat or a combination of these as shown in Fig. 3.4(a).

Rhythm and melody are combined to create complete musical pieces and this observation stands strong for western and most of world music with some exceptions e.g. a musical chant where the focus remains on holding the pitch.

3.1.5 Harmony

Melodies alone can be used to create a musical composition but such compositions may sound empty and disjoint as only a single note is being played at a time. A harmony provides a way of interaction for melodic notes which makes the composition rich and gives it more expression. A harmony is created by playing more than one notes at a given time, and usually is in the background, giving the melody more prominence. There are two main kinds of chords i.e. major and minor. Typically, major chords sound bright and happy, while minor chords sound duller and more mellow. For the key of C, these are formed as shown in the image 3.4 (b). Extended notes including 4th, 6th and 9th are also used within compositions

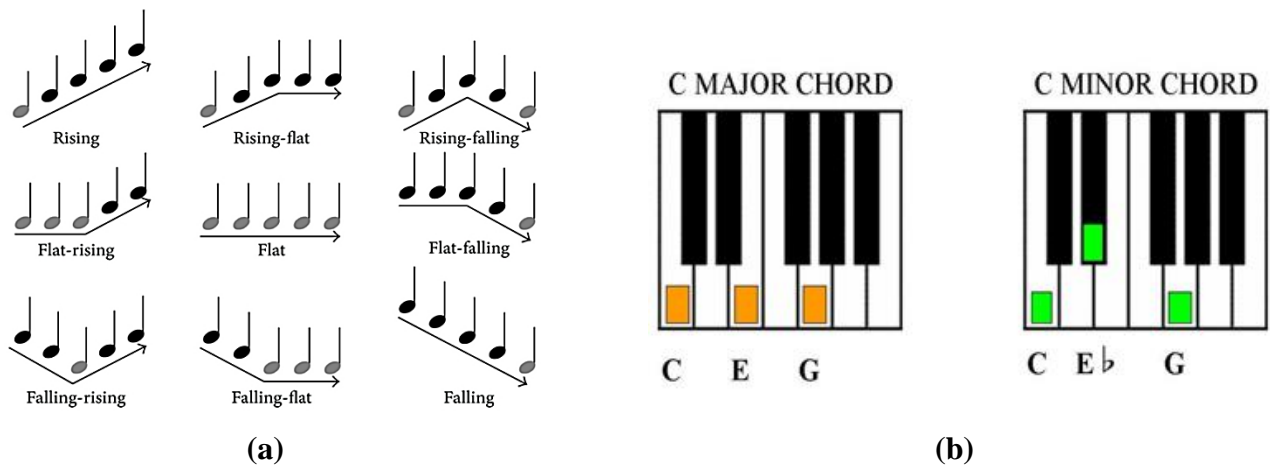


Fig 3.4 (a) melodic notes are arranged as rising, falling, flat or a combination of these to create higher expression. (b) Major and minor chords shown in the key of C.

3.1.6 Long-term Structure

The *global structure* or the *long-term structure* of musical compositions can be split into some wide categories. Western music generally follows a similar structure and generally include following main components defining its global structure:

The *Introduction* of the composition which serves the purpose of establishing the mood of the piece, in terms of scale, tempo and rhythm etc. The *Verse*, which is the start of the musical content and followed by a chorus or a pre-chorus. The *chorus* is the recurring part of the song, often referred to as the *hook*. This is the part the piece is usually known by, and is meant to define the whole piece, as the other sections build up to it. The *conclusion* ends the composition. It can be characterized by a decrease in volume and/or a slowing down of tempo. The conclusion only occurs once and signifies the end to the piece, adding a sense of closure even before the song ends, due to the turnaround in chord progression.

3.2 LANGUAGE MODELLING

Traditionally, language and music have been considered as different psychological faculties by researchers. This view has recently changed with the advent of sophisticated MRI techniques and field of neuroscience. Research in these areas show commonalities in the brain functions associated with both tasks and “*several neural modules are similarly involved in speech and music*” (Tallal & Gaab, 2006).

The problem for predicting the probability of a musical event occurring can be titled as a *Sequence Learning Problem* as highlighted in (Dietterich & Michalski, 1986). This explains music as a set of events with each event having a finite number of features and feature is drawn from a *vocabulary* of finite elements (notes), very similar to a language syntax. By implication of the similarities between structure and neural processing of music and language, *Natural Language Processing* (NLP) techniques can be used to represent music within a computational framework.

3.2.1 Music Language Modelling

In NLP, a *language model* is described as probability distribution over strings of text as shown in Equation 3.1.

$$P(\mathbf{w}_1, \dots, \mathbf{w}_m) = \prod_{i=1}^m P(\mathbf{w}_i | \mathbf{w}_1, \dots, \mathbf{w}_{i-1}) \quad \text{Equation 3.1}$$

It describes the likelihood of a string \mathbf{w}_i from a finite *vocabulary* \mathbf{m} , given all previous strings $\mathbf{w}_1, \dots, \mathbf{w}_{i-1}$. In a *Music Language Model* (MLM), language modelling approach can be used to explain a prior probability distribution (Sigitia, Benetos & Dixon, 2016). The musical sequence \mathbf{y} at time t can be represented as a high dimensional binary vector. The notes being played are represented by turning the binary bits to “on” state at time t . For monophonic music, the problem gets simplified as the input vector is highly sparse, i.e. only one note is played at a time. This allows the independence assumption for musical notes in a single input. Monophonic music, in a language modelling framework can be easily handled using *one-hot encoding* mechanism. Polyphonic music, however, may require more attention as multiple notes being played at a given point in time may be harmonically related.

Recurrent neural networks map an input sequence \mathbf{x} to an output sequence \mathbf{y} at each time step t in order to predict the conditional distribution $P(\mathbf{y}|\mathbf{x})$. For musical sequences, this approach can be used to define the probability distribution over a sequence \mathbf{x} by using the predictions from previous time step $t-1$ as inputs to the next time step t . Outputs at time t will be used as inputs at time $t+1$ in a similar fashion.

3.3 MUSIC REPRESENTATION

Different audio representations have been used in deep learning experiments for model input, processing and generation. These include Raw audio signal representation and symbolic music formats.

Raw audio is used by (Sarroff & Case, 2014) as an input to their neural model to perform audio synthesis and generate music in real time. *WaveNet*, a generative model for raw audio by Google Brain uses such audio data as input to their generative neural model to mimic human speech as shown in (Van Den Oord et al., 2016). *Nsynth*¹, a deep neural synthesizer developed by Google Magenta project, performs granular and additive synthesis by directly learning and generating audio samples by performing frequency manipulation in a deep neural network. However, due to heavy computational cost involved in training, and generating with raw audio signals, most of the experiments in deep learning for music generation tend to use symbolic musical representations like ABC format or MIDI.

Due to author’s previous familiarity with generating and processing MIDI data in studio and production environments, it was decided to select this format for representing musical data as

¹ NSynth, Neural Audio Synthesis (<https://magenta.tensorflow.org/nsynth>)

model inputs and outputs. This decision was informed through literature review and also considering the available computational infrastructure.

3.4 MIDI – MUSIC INSTRUMENTS DIGITAL INTERFACE

MIDI solves the problem of representing sheet music digitally in modern music production systems for capturing performances, arranging tracks and communications between musical devices and application. MIDI data is composed of MIDI messages for musical events taking place within a musical piece at discrete time steps instead of audible music as in raw audio signals. A MIDI track is thus a sequence of time ordered events. MIDI messages contain music meta information (tempo, time signature etc.), pitch, velocity and dynamics control (vibrato, pitch bend etc.) of a note at a given time step. MIDI data is played back using a *MIDI synthesizer* which may be used to further design and modify the output sound. A “*Note on*” events indicate that a specific note is played at a given time step. A “*Note off*” event, on the other hand indicates when a musical note ends. This allows easy extraction time-ordered note placement information from MIDI music as required by the proposed experiments.

Example of a note on event is **<Note on, 0, 60, 50>** which explains that for channel 1 (0 refers to channel 1), start playing a middle C (MIDI note 60) with velocity 50 (where velocity range is 0-127). A note off messages can be shown as **<Note off, 0, 60, 20>** which means that using channel 0, middle C note should stop playing with a velocity of 20.

3.4.1 Time Discretization:

Representing musical events in a computational model depends heavily on effective representation of time due to the time-series nature of musical sequences. In order to process the MIDI data within a language modelling framework, global time discretization/quantization is required to temporally define the musical events taking place within the sequence. (Eck and Schimdhuber, 2002) first identified a strategy to achieve this objective with RNNs. Generally, a time step is set to the smallest note duration found within the dataset i.e. a 16th or an 8th note. It is a very simple approach in terms of implementation, however, may prove to be computationally expensive for long sequences showing minimal note transitions. This approach works independently of the duration of actual notes.

3.5 THE PIANO-ROLL REPRESENTATION

The idea of piano-roll representation of a musical piece, either monophonic or polyphonic, has been borrowed from automated and self-playing pianos (also known as pianola or player piano) which appeared in early twentieth century.

In most modern digital music production setups, relying heavily on DAWs (Digital Audio Workstations), the term piano roll is used to display and edit MIDI data in a user friendly graphical manner. The x axis in a typical piano roll represents time where the y axis represents the pitch the note being played as shown in Fig. 3.5.

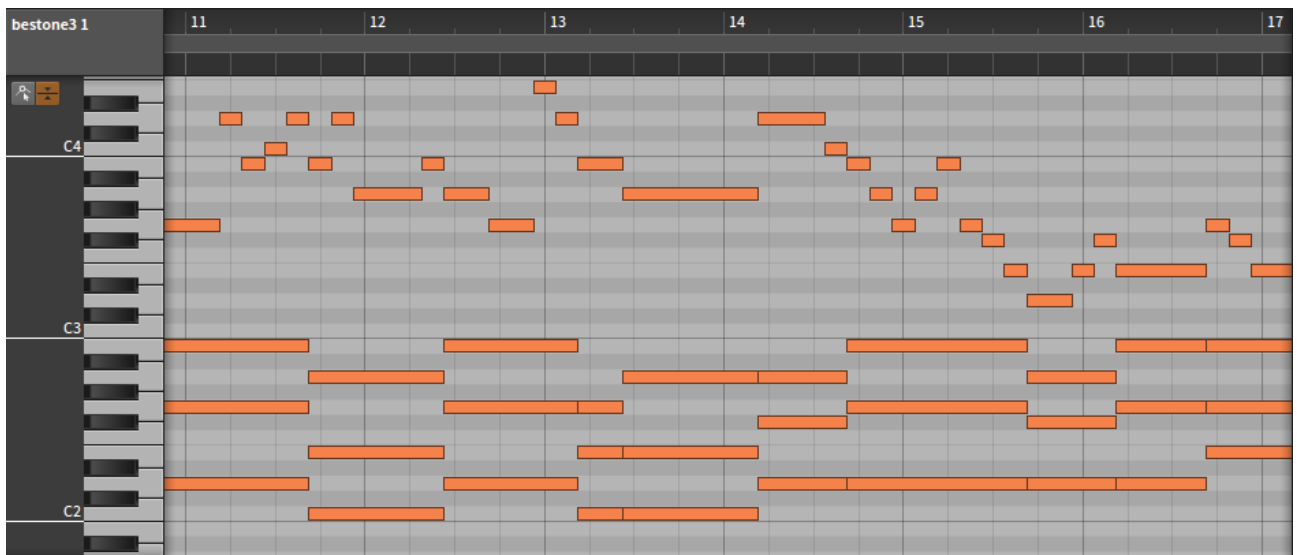


Fig 3.5 A piano roll representation of a MIDI file within a Digital Audio Workstation

Piano-roll is a very frequently used representation technique despite having some limitations. One key limitation of piano roll representation is inability to represent a note off event. As a result, this representation cannot distinguish between a sustained note and notes being played repeatedly with same pitch. More complex representations can be used to represent an actual note duration and dynamics within computational frameworks.

Literature review revealed that one-hot encoding is the most widely used input encoding mechanism used with monophonic symbolic musical data in RNNs due to its simplicity and effectiveness, especially for monophonic data. One-hot encoding can be used to generate monophonic melodic and harmonic notes by converting harmony notes to a chord class.

Chapter 4: METHODS AND APPROACHES

In literature review, key areas identified towards developing a research and development framework in the area of generative music were *Data Representation and Encoding, Feature Engineering, Network Architecture and Generation Strategy*. This section will highlight each of these areas with methods and approaches chosen to fulfil stated research objectives.

4.1 SUPPORTING SOFTWARES AND TECHNOLOGIES

In order to implement the methods and approaches described in this section, following set of tools and technologies were used within this project.

- **TensorFlow Deep Learning Framework** (<http://www.tensorflow.org>)
TensorFlow 0.8 on Linux was used to build and train neural networks for melodic generation. This learning environment was selected because a vast number of current research activities into the field of music generation are being conducted using this framework which provides a great source for learning and experimenting with latest developments into generative music.
- **Python Programming Environment** (<https://www.anaconda.org>)
Continuum Analytics Anaconda based distribution of Python programming language was used in this experiment. Python 2.7 was selected due to large community support and seamless integration with all major Python libraries.
- **Python-MIDI Toolkit** (<https://github.com/vishnubob/python-midi>)
Python-midi library was used for parsing musical sequences. This library has been used in a number of music analysis and generation experiments to parse MIDI data prepare input encoding for further processing.
- **Mingus** (<https://bspaans.github.io/python-mingus>)
Mingus is a music theory and notation library for python and it was used for converting harmonic notes to chord classes along with Python-midi library for data pre-processing.
- **Bitwig Studio 2** (<https://www.bitwig.com>)
Bitwig studio is a multi-platform Digital Audio Workstation (DAW). Bitwig was be used in proposed experiment for inspecting generated midi files, generating audio from midi and to conduct listening and comparisons between different generative experiments.

4.2 IDENTIFYING A CODE BASE

As our initial research question and included objectives have not been directly addressed in any previous research, it was decided to use set up a baseline model first to compare the performance improvement in following augmentation related experiments. The base model set for the was derived from an online experiment by Yoav Zimmerman, titled “*A Dual Classification Approach to Music Language modelling*” (Yoav, 2016). This experiment uses the *Nottingham Folk Music Dataset* to train an RNN architecture with a novel loss function taking into individual melody and harmony softmax losses before calculating the final loss value. The architectural overview of this experiment is shown in Fig. 4.1.

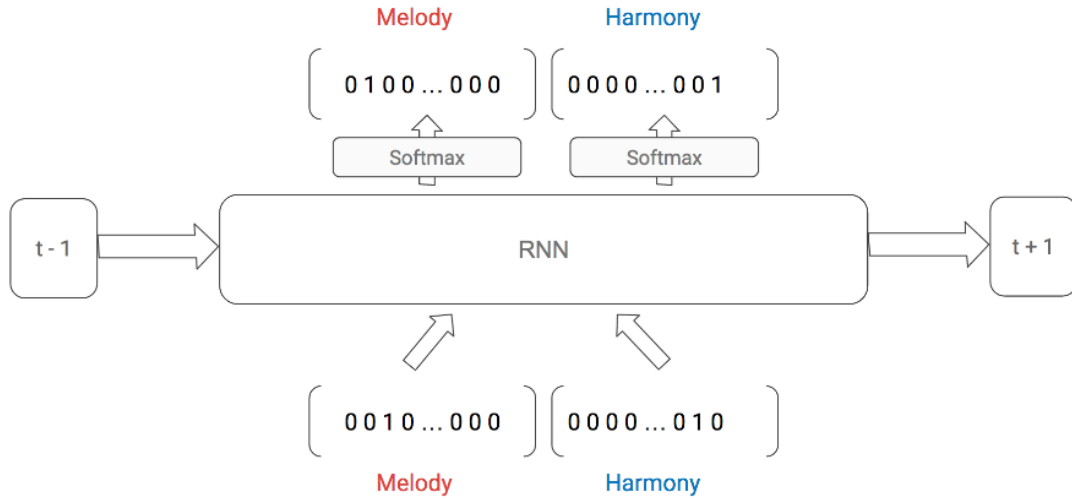


Fig. 4.1 The Dual Softmax Classification model by Yoav Zimmerman (Yoav, 2016)

This code base parses MIDI sequences with a *monophonic* assumption for melody i.e. only one note playing at a given time step. It also assumes that harmony notes at each time step can be classified into as chord class which belongs to a finite chord dictionary containing major and minor chords for each note i.e. maximum 24 classes. These assumptions were made for the sake of simplicity towards music language modelling, and were maintained for proposed augmentation of engineered features and other related experimentation.

This code provided with this experiment was chosen for base model development as it fulfilled a number of basic requirements that were found helpful towards proposed framework development, as summarized below:

- The experiment uses symbolic music data representation (MIDI) within the training and generation phases, thus making it possible to conduct the proposed feature augmentation and other experiments without incurring a heavy computation cost as compared to processing raw audio.
- The dataset used in this experiment includes songs with a variety of musical styles. The compact size and variety of data in this dataset was considered suitable towards model development as it required minimum computational cost and offered standard approaches towards data pre-processing and representation.
- The model uses a dual-softmax approach which calculates the harmony and melody loss independently from each other in order to calculate the total loss at each time step. This showed a potential towards future experiments to learn the impact of augmented features on melody and harmony separately.
- The experiment uses Neural Language Modelling techniques with RNN architecture for predicting the melody and harmony notes as required by proposed experiments (with options to use vanilla RNNs or GRUs for comparison).
- The experiment uses standard open source development tools including Python, Numpy, Python-midi and TensorFlow. These tools are widely used in the domain of deep learning and generation with good community support.

The code base for this model provided a great template for understanding different components of a deep learning framework including control over training process and learning parameters, a flexible generation and pre-processing framework with room for further modifications, automatic logging of training process and a parametric grid search.

4.2.1 Changes Made To The Code Base

Upon Investigation, and as mentioned by the author on the GitHub repository¹, some shortcomings in the code base were identified. Following changes were made to this model and the accompanying dataset to create the base model for our experiments and to compare the results of later experimentation.

- **Chord Classes.** The code to identify and create one-hot vectors for harmonic sequences was producing 33 different chord types whereas a maximum of 24 chords (major and minor for each key, for 12 keys in total) were expected. Upon investigation, it was realised that while using python Mingus library, some chords were being labelled in non-uniform manner hence generating extra classes of chords. This error was fixed in the code base and a total of 23 chords were identified as shown in Fig 4.7. *The `Resolve_Chord()` function shown in **Appendix C.1** describes this step.*
- **Early Stopping Criteria.** The code used Dropout as the only way of regularisation. This required running the code to a fixed number of epochs during the learning phase even if the model's performance was not improving. This also caused model to overfit and proved non-productive during experimentation. The training section of the code was hence modified to apply *Early Stopping Criteria* (Stop training if no improvement in validation loss is seen for 15 epochs) as an additional regularization method. This addition can be seen in "`train.py`" in **Appendix C.2**.
- **Variable Length Sequences.** The language model implemented within the code only catered for fixed length sequences. In order to process the variable length MIDI sequences found in the dataset, the code was initially truncating the sequences to make them a multiple of 128 which is chosen mini-batch length. Therefore, information about the structure of the midi sequences was being lost which was considered to have negative impact on proposed experiments. This shortcoming was fulfilled by zero-padding the sequences to preserve their length with a counter pointing out towards the end of sequence and disregarding any sequences of all zeros following that. Zero-padding approach has been highlighted in **Appendix C.2**, in program's main body.
- **Data Split.** The data was found to be split three ways as testing, training and validation data, but no cross validation was applied. This reduced further reduced the size of an already compact dataset. As cross validation was not planned at this stage, the data was re-split as test and train data for experiments, while ensuring a balanced distribution of musical styles based on MIDI filenames.
- **Model Architecture.** The Model used same number of inputs and outputs during training and generation phases. For feature augmentation experiments, the model was modified and made more flexible to allow the change in the size of output layer to suit

¹ https://github.com/yoavz/music_rnn

the requirements of particular experiments. The code for developing model architecture with modifications is detailed in “*model.py*”, attached **as Appendix C.3**.

4.3 EXPLORATORY DATA ANALYSIS

The code base uses Nottingham Dataset¹ for training and validation. In order to get an insight into the dataset, an initial exploratory data analysis was performed. This dataset offers a collection of over a thousand folk tunes from Britain and America. MIDI Files in the dataset contain 3 tracks with track 0 containing song meta information, track 1 and track 2 with temporally relevant musical notes i.e. melodies in track 1 correspond directly to harmony data in track 2. The distribution of musical styles found within the dataset are shown in Fig. 4.2.

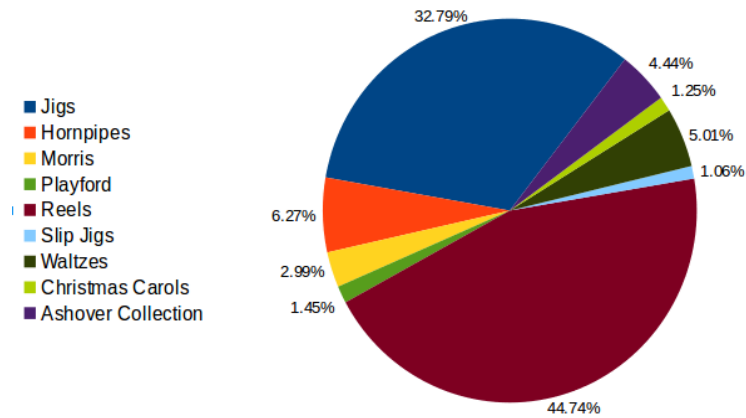


Fig. 4.2 Style distribution in Nottingham dataset.

Fig. 4.2 highlights the imbalanced distribution of styles within the dataset with Jigs and Reels (folk dance music mainly from England and Europe) making up more than 75% of the dataset. Also, songs in the dataset are of different time signatures with 4/4 (547 songs) and 6/8 (355 songs) being most dominant classes as shown in Fig. 4.3

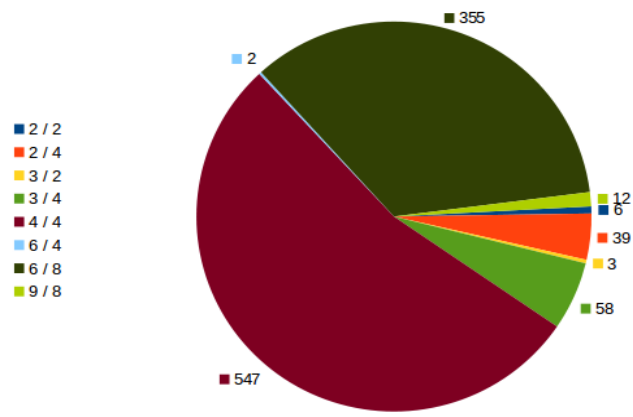


Fig. 4.3 Time signature distribution in Nottingham dataset.

¹ <http://abc.sourceforge.net/NMD>

As the proposed experiment aimed at investigating the impact of structural features on learning and generation in general, instead of learning and translating styles, it was decided to further the experiments with this dataset.

4.4 DATA PARSING AND REPRESENTATION

One of the important aspects to be considered while analysing and generating musical sequences is selecting a suitable way to represent musical content within the computational frameworks. This decision may highly impact how training and generation phases are conducted and also the quality of generated output. To convert continuous music into notes being played at discrete time intervals, as required for language modelling approaches, the code base uses a time step of 1/16th of complete note length based on MIDI resolution. Using this time step, pre-processing stage uses *Python-midi* library to parse track 1 for learning melody note transition probabilities, melodic scales and patterns within the song rhythm and track 2 for extracting relevant harmonic content. Track 0 is parsed for sequence meta information as shown in Fig. 4.4.

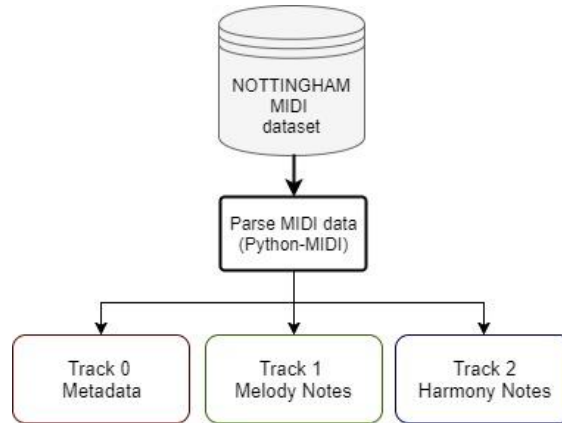


Fig 4.4 Python-midi library is used to parse MIDI files with three tracks including metadata, melody and harmony notes

An example of information extracted from track 0 is shown in Fig 4.5.

Key	Type	Size	Value
name	str	1	reels_simple_chords_173
path	str	1	data/Nottingham/train/reels_simple_chords_173.mid
signature	tuple	2	(4, 4)
ticks_per_quarter_note	int	1	48

Fig 4.5 Song metadata contained within track 0 of MIDI files

Melody and harmony ranges used in this scheme were found to be best suited for parsing Nottingham dataset and were left unchanged as shown in **Appendix B.1**. These ranges can be flexibly changed to meet the needs of other datasets which may show activity outside the defined ranges.

4.4.1 Melody Encoding

The code base maps track 1 data as a discretized one-hot vector with the specified time step and melody range. A simple example is shown in Fig. 4.6. A 35 bit long one hot vector is created at each time step using Python-midi and NumPy libraries. The parsing is performed with a predefined time step of 1/16 of a bar (1/4th of quarter beat in 4/4).

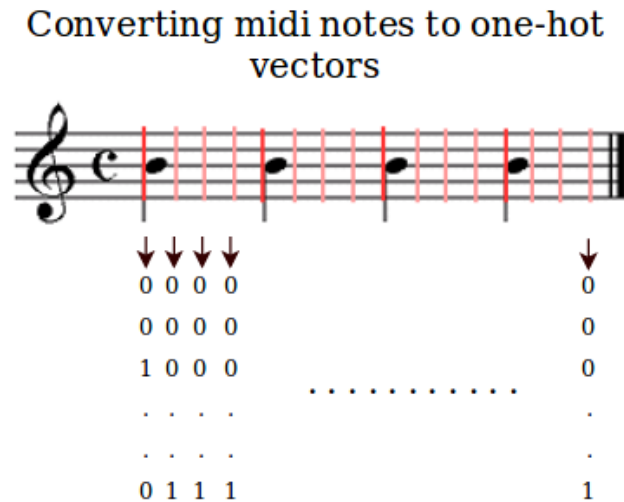


Fig 4.6 This simplified example described parsing of melody track with discretized values taken at 1/16th of a measure (1/4th of a quarter beat in 4/4 format).

4.4.2 Harmony Encoding

In order to convert harmonic notes into their respective chords, the code base uses Python *Mingus* library to determine the chord being played. The base model converts extended chords including 6th, 7th, 9th, 11th and diminished into respective major and minor chords for simplicity. Some minor corrections were made to this part of the code base as described earlier resulting as 22 chord classes shown below, plus one class for unknown chords that could not be identified.

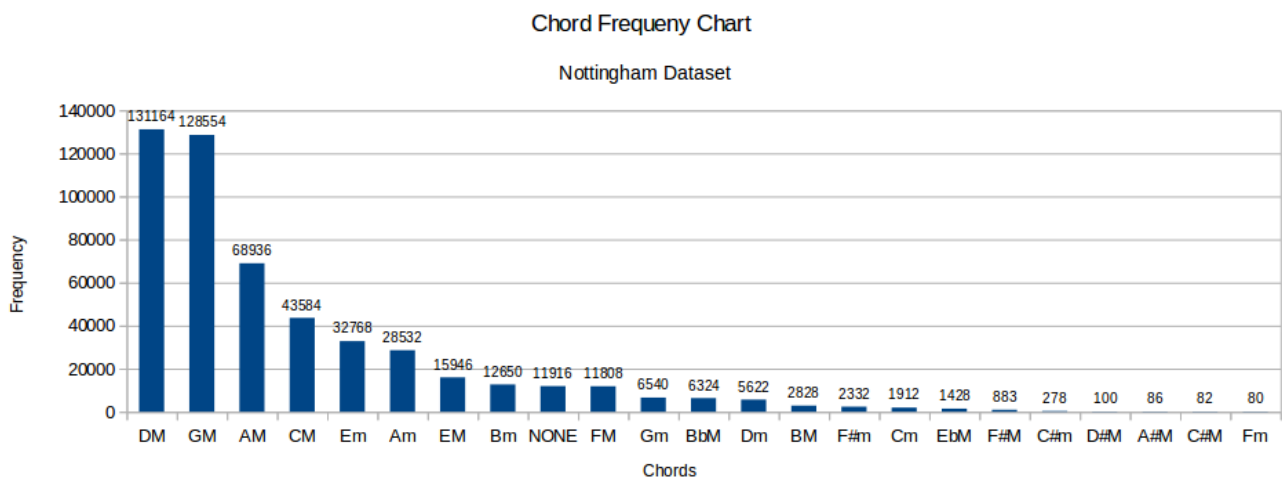


Fig 4.7 A chord frequency chart highlighting frequency of 22 chord classes found in Nottingham data set after making modifications to the code base.

A *chord index* is created for all chords found within the corpus and a one-hot harmony vector is generated from this index at each time step, in a similar fashion as melody encoding mentioned previously. The process of converting notes to chords, simplifying them to respective Major and Minor types and creating a harmony one-hot vector based on chord index is given as a flowchart in **Appendix B.2**.

4.4.3 Input Encoding

The MIDI to piano roll mappings were created by appending melody and harmony one hot vectors on each time step and thus creating a final melody encoding. Fig 4.8 shows an example of this encoding of one of the input files.

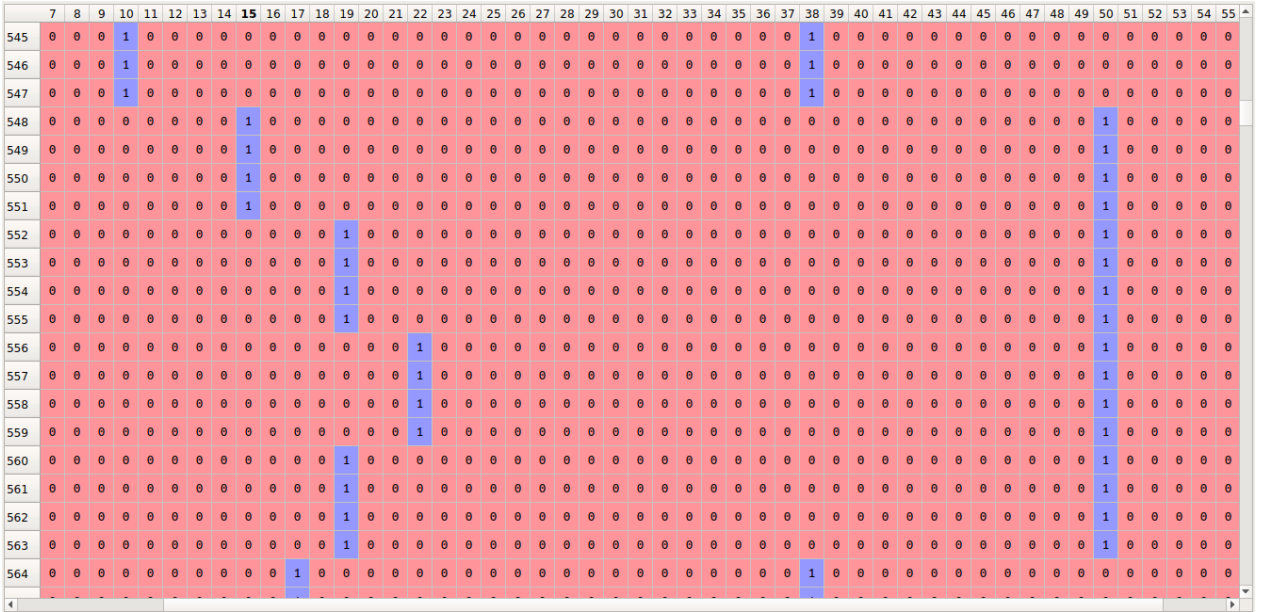


Fig. 4.8 An example input encoding (0- 35) melody, (36 - 57) harmony

The parsing conditions and assertions made in this code resulted in a loss of about 16% of the data (over 200 tracks), with the final dataset having 997 sequences, split as 70/30 train/validation dataset.

The above section describes the steps taken to create the baseline model. Results in section 5.1 shows a comparison between the original experiment and baseline model with modifications.

Following sections will highlight feature augmentation approaches performed to answer the research objectives of this project. The steps are highlighted in Fig. 4.9 and the code is provided in **Appendix C.1**.

4.5 AUGMENTATION OF ENGINEERED FEATURES

For proposed Structure Augmented LSTM (StructLSTM) model, structural information as extra input features was appended to input encoding created for baseline model. A similar approach for feature augmentation has been tried with *Lookback RNN* architecture with google Magenta

(Waite, 2016), that uses a one-hot vector containing information from previous events while predicting next musical event.

In order for the neural network to learn the extra structural knowledge of the input MIDI files, a framework was developed to extract required information from the corpus, calculate and augment custom features to input encoding of the base model. These extra features were meant to provide extra information about the *duration*, *metre* and *rhythm* of the MIDI sequence to allow the model to learn some extra information about these structural elements during the training phase. In the generation phase, a set of similar features were calculated and introduced into the generative process using a feature generator.

4.5.1 Augmenting Song Duration Counter

The base model learns short term association between melody and harmony notes played within the training data, but lacked the ability to conclude a song in a human-like fashion. As a result, the generated melodies end abruptly and at random.

In order to address this problem, a simple duration counter was calculated and appended to input encoding in an attempt to add extra information about temporal and structural associations between melodies and harmonies played in song introduction and conclusion. The counter values were set to start with a value equal to number of total time steps in the sequence, and decrement with each step until the last time step, where counter value reaches zero. In other words, it “counts down” to the song conclusion. This augmentation was performed with a motivation that intro and outro of western music normally consist of chorus and are generally very similar in terms of structure and notes played. An LSTM, by default, cannot relate these temporally distant parts of the song. The proposed countdown augmentation was performed with an intent to help model understand the relationship between the beginning and the end of the song by learning that the melodies and harmonies played at the end, as the counter approaches zero, are similar to those played at the very beginning, when the counter value was at maximum. The countdown augmentation was also implemented in the generation phase to help model relate the beginning of the song to the end, while providing it with an engineered counter input based on the length defined for generated music.

4.5.2 Counter Modifications

A number of further modifications were applied to the counter feature mentioned earlier to observe the impact on model behaviour under different encodings. Counter input was normalized in order to allow faster training. The effect of normalisation on neural networks has been discussed in detail by (Krizhevsky, Sutskever and Hinton, 2012) and (Ioffe and Szegedy, 2015). Following this idea, countdown values were normalised from 1 to 0 for all sequences in the dataset. The results of this experiment are given in detail in section 5.2.5.

As a further refinement to the previous augmentation approach, the counter was rolled back so that zero value pointed to the beginning of last note rather than the last time step in the MIDI sequences. This was from the motivation that the last note in a song may be played for shorter

or longer periods of time and hence had an impact the total number of time steps in the MIDI file. As in western music, a song usually ends on a strong beat, rolling the countdown back to the time step where the last note is played might allow the model to better conclude the musical sequence by trying to learn and repeat the similarity between maximum and minimum counter values seen in the training dataset. A simple routine written in python fulfilled this purpose.

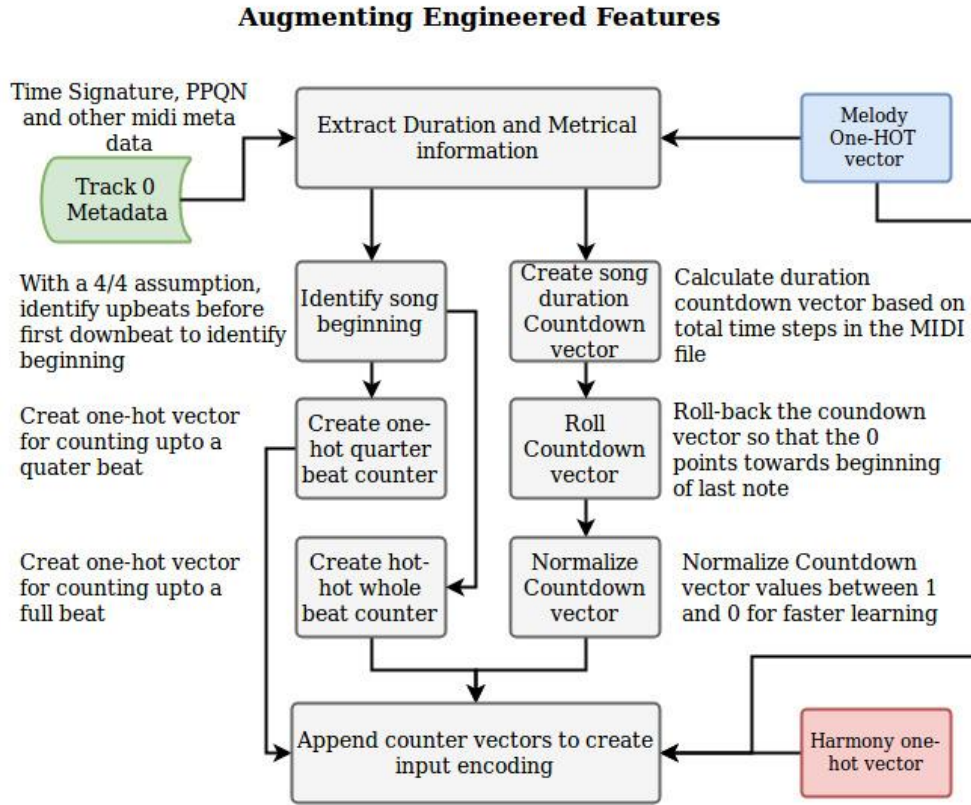


Fig. 4.9 Feature Engineering and Augmentation to the input encoding. This framework allows to create one-hot vectors for counting any beat duration. Initial setup was used to create one-hot vectors for counting a quarter beat (4 time steps), and a whole bar (16 time steps). The framework is flexible enough to modify for further experimentation.

4.5.3 Augmenting Metrical Information

The song metre, associated with recurring patterns of note combinations in musical bars are known to have a strong impact on listener's perception of the music being played. According to Justin London, the metre of a musical piece builds our initial perception as well as future anticipation of musical patterns *"that we abstract from the rhythm surface of the music as it unfolds in time"* (London, 2012). A musical bar is *"foundation of human instinctive musical participation"* as stated by (Scholls, 1977).

Following this motivation and in order to further the augmentation framework, extra metrical information was augmented to the input encoding with a 4/4 assumption. This involved creating two one hot vectors, first to count each time step towards a quarter beat (4 time steps),

and a second counter to count quarter beats up to whole bar (16 time steps or 4 quarter beats). This pattern was repeated till the end of sequence as shown in Fig 4.10.

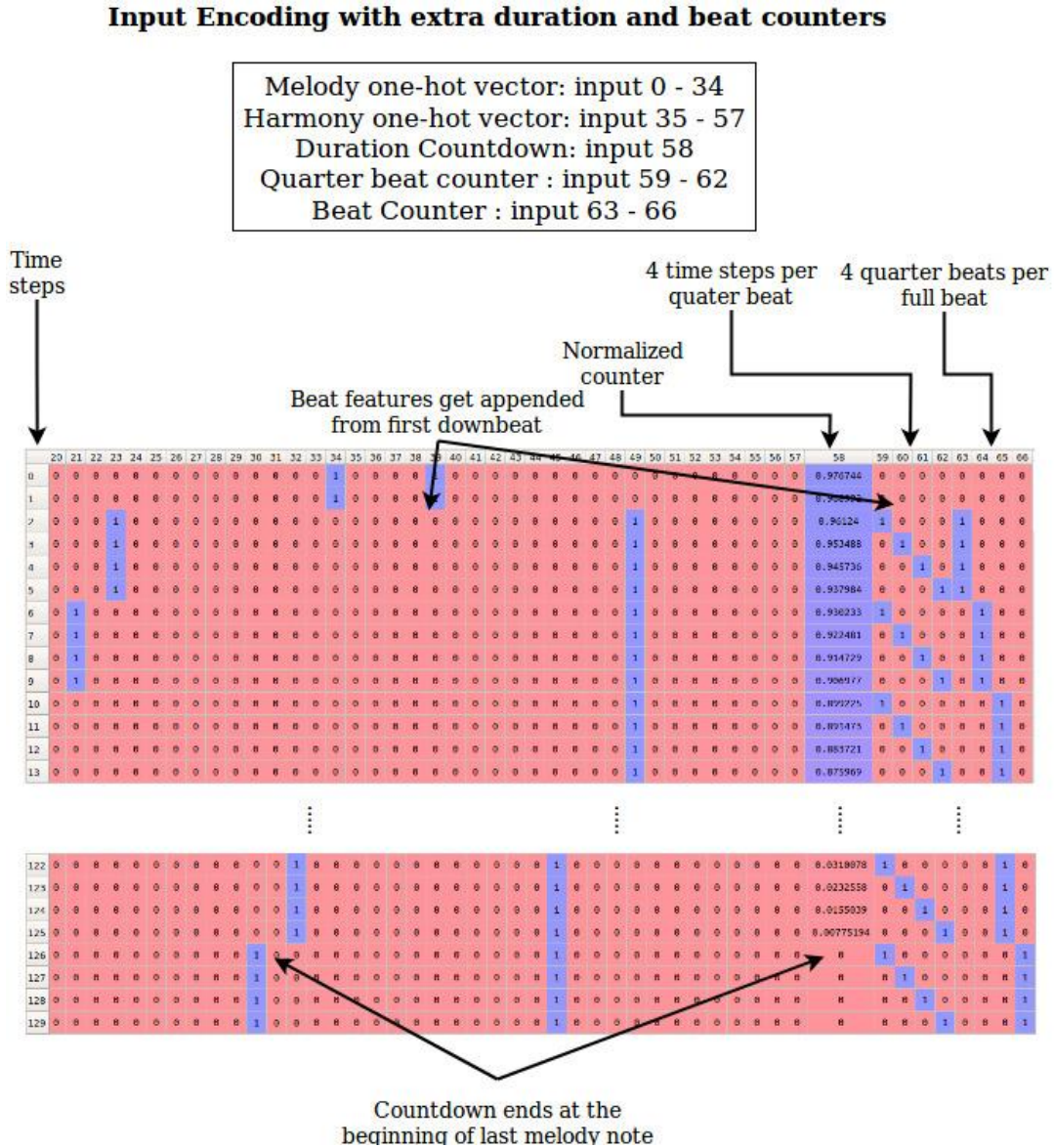


Fig 4.10 Input encoding after duration and metrical engineered feature augmentation.

During initial listening tests with the dataset, it was identified that a large number of songs start from unstressed notes before beginning of first bar. This is known as *Anacrusis* and could lead to augmentation of incorrect metrical values. In order to associate metrical information with relevant notes, a simple routine was devised (with a 4/4 assumption) that counted the total number of time steps in the musical piece and performed integer and modulo divisions by 4 on the sequence length to identify total number of quarter beats and bars in the track. The results of modulo division were considered time steps before the first bar and metrical information was appended starting from first known downbeat. Although this was a naive implementation to accommodate anacrusis, but it was found that it performed the required function correctly for 4/4 MIDI files according to set assumptions. The metrical duration and metrical engineered

feature augmentation approach has shown in Fig. 4.9. The effect of this approach on final input encoding has been shown Fig. 4.10

*The python code for pre-processing and augmentation steps described in section 4.5 has been presented in this report in **Appendix C**. The file “**proprocess.py**” in **Appendix C.1** shows above steps with detailed annotations.*

4.6 MINI-BATCHING

In order to preserve the duration of sequences in Nottingham dataset, all sequences were zero-padded to make them a multiple of mini-batch length of 128. The code base from (Yoav, 2016) used sequence truncation to cater for mini-batching where all mini-batches must be of the same length. Zero padding allowed sequences to retain their duration.

TensorFlow deep learning framework uses tensors for input and computation of data. a *Tensor* object in TensorFlow is a generalization of high dimensional matrices and vectors and is represented within computation as an n-dimensional arrays, described by its data type and dimensions or *shape*. The mini-batching process offered by the code base was modified to cater for different input and output dimensions as a result of feature augmentation. The process includes following key steps.

1. Zero-padded sequences are stacked as tensors using NumPy array stacking and axes manipulation techniques.
2. Sequences are replicated as input and target data with target data rolled back to a single time step to be identifies as the ground truth.
3. Augmented features were removed from the target dataset as these features are not required in the output.
4. Target one-hot features are converted to class labels for notes and harmonies in the input encoding so that each melody and harmony label is between $[0, \text{num_classes}-1]$. This allowed TensorFlow to calculate probability of a given label in an exclusive manner. TensorFlow uses *tf.nn.sparse_softmax_cross_entropy_with_logits* in order to classify data into hard classes which was a requirement for planned language modelling setup.
5. Input examples and labels created in the last step were split into mini-batches based on the mini-batch length of 128 time steps i.e. 8 bars of music. This defined the number of unrolling steps for backpropagation through time.

These steps allowed to encode data into a format and structure that was suitable for experimentation in TensorFlow deep learning framework. The process of creating tensors is summarized in Fig. 4.11.

*The code for above steps can be viewed in functions provided in **Appendix C.2**, “**train.py**” script.*

After creating the input encoding and mini-batching data, it was fed to the network with a flexible architecture to run different layer and layer size combinations. Following the original experiment and as seen in other similar experiments during literature review, the experiments

were tried with and two hidden layers and layer size ranging from 100 – 200 as detailed in the results section.

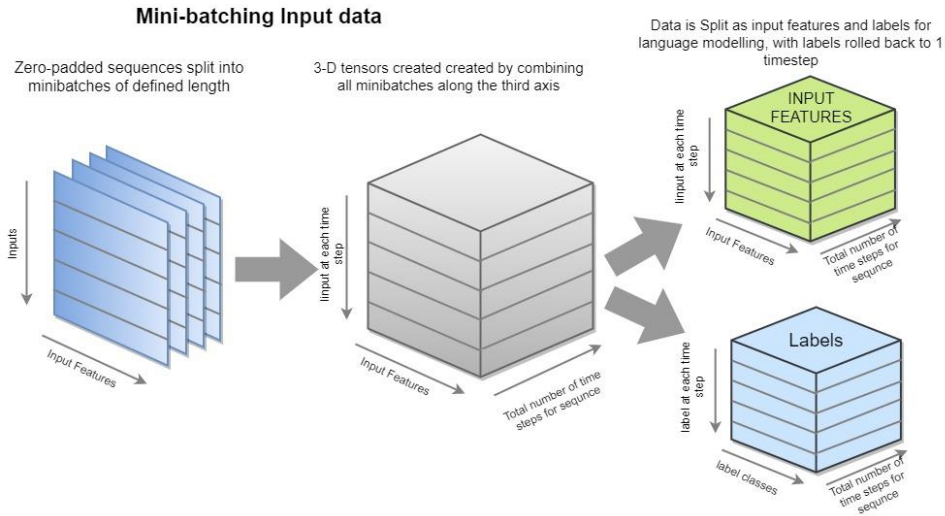


Fig. 4.11 Creating mini-batches from zero padded input sequences. For training in a language modelling framework, the labels were rolled back to a length of 1 time step to be identified as ground truth for data in the input feature tensor.

4.7 StructLSTM (Structure Augmented LSTM)

Selected code base offered experimentation with simple RNNs, GRUs and LSTMs. LSTMs was chosen as an architecture of choice as described in the critical context of this report. Cross Entropy Loss was used to measure error between the network output and predicted values. The model uses Tensorflow's *MultiRNN* class that used same layer size for all the hidden layers. *RMSProp* adaptive learning method, as mentioned by Hinton applied used for backpropagation learning (Tieleman & Hinton, 2012). The resulting network with augmented features was names as *StructLSTM* (Structure Augmented LSTM) in order to differentiate it from original code base and baseline model.

For the loss function of the base model uses a weighted sum of melody and harmony loss as shown by (Yoav, 2016) using Equation 4.1 as given below.

$$L(z, m, h) = \alpha \log \left(\frac{e^{z_m}}{\sum_{n=0}^{M-1} e^{z_n}} \right) + (1 - \alpha) \log \left(\frac{e^{z_{M+h}}}{\sum_{n=M}^{M+H} e^{z_n}} \right) \quad \text{Equation 4.1}$$

Here M and H are melody and harmony class. The function calculates the log loss at a time step for the output layer $z \in \mathbb{R}^{M+H}$, a melody label class m , and target harmony label class h . α , termed as melody coefficient, can be altered to shift the focus of loss function between memory and harmony. For the experimentation, α was set to 0.5, giving equal importance to individual melody and harmony losses.

The model, earlier described as having the same set of input and output features was slightly modified to allow augmentation of extra engineered features into the input and predict the

output for next time step without predicting these added features. The final layer of the network was modified to generate the melody and harmony probabilities only, while learning from input notes and augmented features.

*Using the approaches mentioned above, the final model architecture for StructLSTM, including necessary pre-processing stages has been summarized in Fig. 4.12. **Appendix C** with “**preprocess.py**”, “**train.py**” and “**model.py**” file provides the python code for this architecture in TensorFlow.*

StructLSTM (Structure Augmented LSTM) Architecture

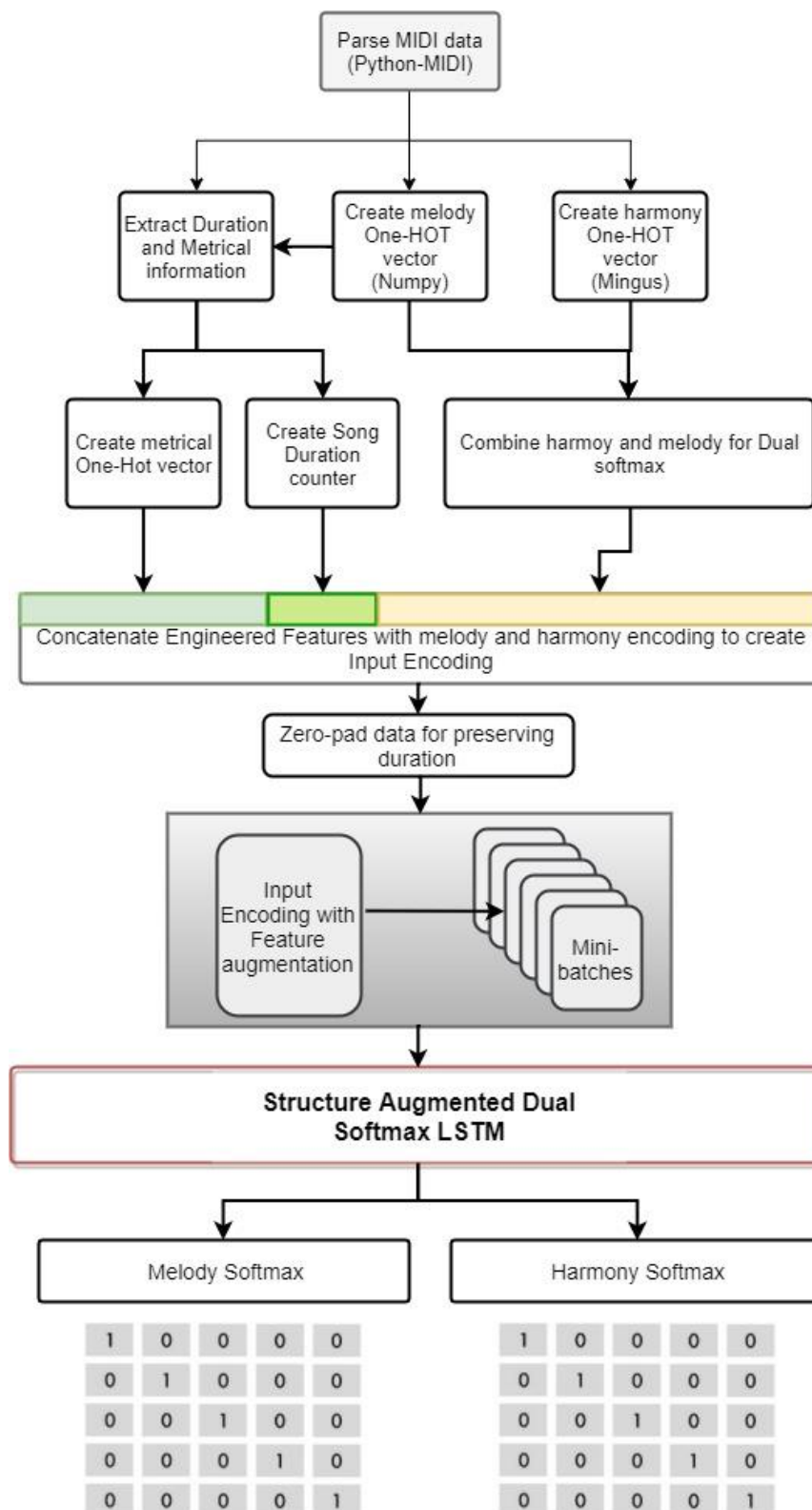


Fig. 4.12 The proposed StructLSTM framework including pre-processing, feature engineering, feature augmentation and network training stages. **Appendix C** provides the code modified and developed to setup this architecture.

4.8 GENERATION STRATEGY

The generative strategy was built upon the generation framework developed by (Yoav, 2006) which uses *conditioning* and *sampling* techniques to generate new sequences (Briot & Patchet, 2017) (Karpathy, 2015). In order to generate new sequences from the trained model, some changes were made to the original experiment to allow creation and augmentation of the engineered features, similar to the ones seen during the training phase.

A feature generator was programmed into the generation stage of baseline model as shown in Fig. 4.13. Based on the total number of sampling steps defined, a countdown vector was calculated and appended to the input sequences during conditioning and generation phases to identify introduction and conclusion of the generated sequence, similar to sequences in training dataset. Metrical vectors, with a 4/4 assumption were also created and augmented to the generated sequence for each time step in a similar fashion. This strategy was developed to view the impact of engineered features on the generative process. A number of experiments were performed as detailed in Results chapter. The feature generator was developed in a flexible way to allow further augmentation of similar engineered features for further experimentation.

*The code for generation using conditioning and sampling is provided as **Appendix C.4** as “**structLSTM.py**”.*

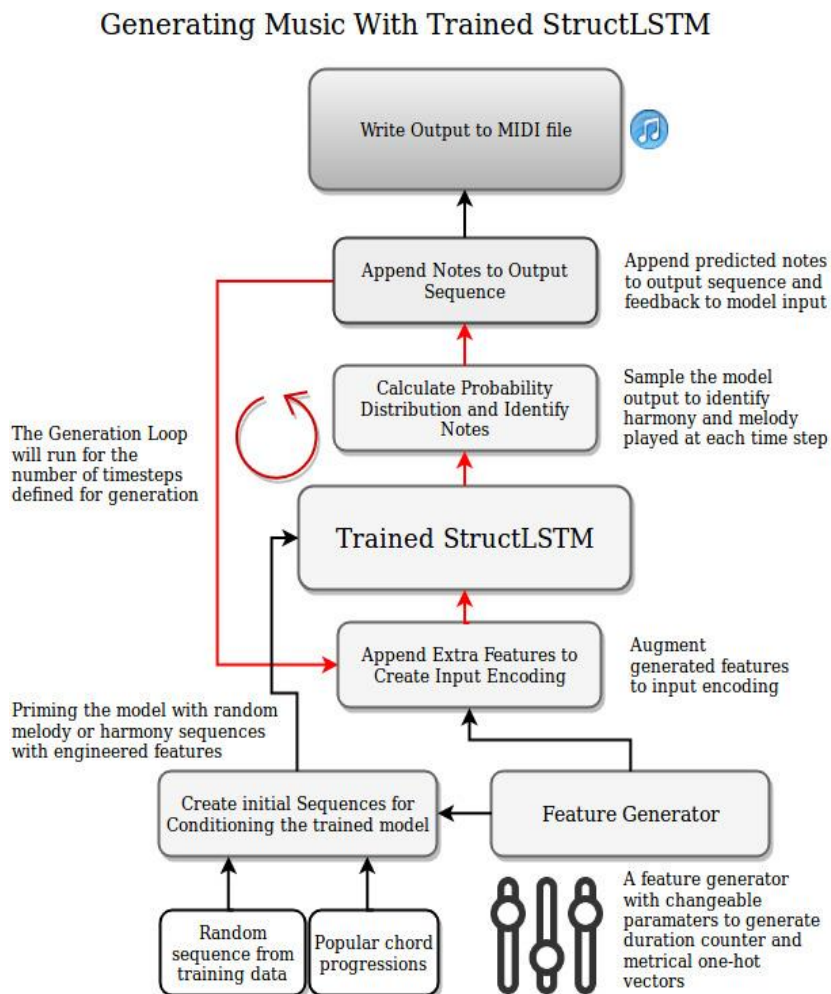


Fig. 4.13 The generation framework. A feature generator is introduced in the generative process to calculate and append engineered features for conditioning and sampling from the output

Chapter 5. EXPERIMENTS AND RESULTS

With the experimentation framework described in the previous section, a number of experiments were conducted with different feature augmentations. For each experiment, a grid search was performed to measure the impact of implemented changes on model behaviour. A number of musical sequences were generated and inspected for each experiment. A number of piano roll representations were also generated to visually analyse melodic and harmonic structure of the sequences.

5.1 SETTING UP THE BASE MODEL

The re-splitting of dataset and changes made to the code base, as listed in section 4.2 of this report required setting up a new baseline model. The first experiment was conducted by training the modified dual soft-max RNN architecture without any feature augmentation, and generating musical sequences. A grid search was applied over following parameter values.

Drop Out: [1, 0.75, 0.5]

Number of Hidden Layers: [1, 2]

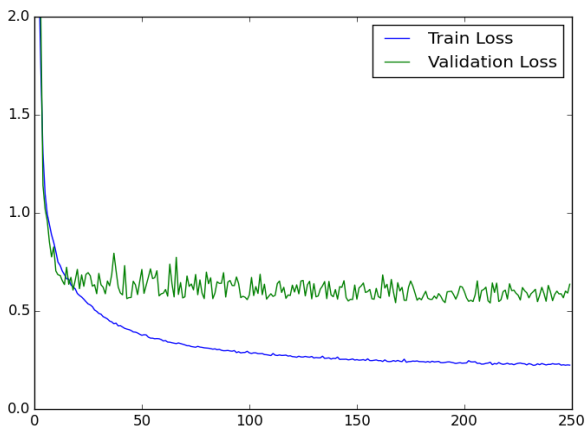
Hidden Layer Size: [100, 150, 200]

An early stopping criterion was applied to training that stopped training a model if no improvement in validation loss was seen for 15 epochs. The results of the above grid search can be viewed in Table 5.1, with best model is highlighted in red.

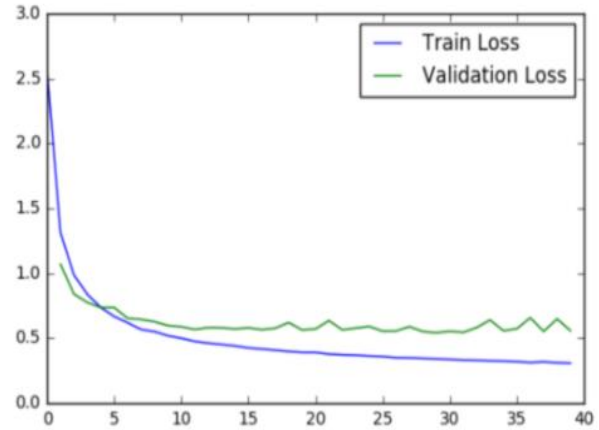
No of Layers	Hidden Size	Drop Out	Train Loss	Valid Loss	Ave. Time/Epoch
1	100	1	0.27369	0.62594	27.64
1	100	0.75	0.33747	0.58984	27.39
1	100	0.5	0.37482	0.56831	27.81
1	150	1	0.21579	0.63369	32.8
1	150	0.75	0.26587	0.57131	33.19
1	150	0.5	0.2863	0.55885	33.47
1	200	1	0.21743	0.63523	43.38
1	200	0.75	0.25078	0.57010	43.07
1	200	0.5	0.29506	0.55469	43.71
2	100	1	0.24358	0.65003	54.84
2	100	0.75	0.27061	0.56347	53.79
2	100	0.5	0.42504	0.57117	57.32
2	150	1	0.20702	0.63256	70.95
2	150	0.75	0.23345	0.56472	72.09
2	150	0.5	0.30345	0.54160	74.48
2	200	1	0.19456	0.63375	95.03
2	200	0.75	0.21508	0.54527	94.83
2	200	0.5	0.33764	0.56715	96.2

Table 5.1 Grid search results for baseline model

The performance of highlighted model from Table 5.1 is shown in Fig. 5.1(b).



a. Model loss reported in (Yoav, 2016)
Valid loss = 0.5363



b. Baseline model with modifications
Train loss = 0.30345, Valid loss = 0.5416

Fig 5.1 The training and validation loss for original experiment is shown in (a). The performance of the baseline model is given in (b), showing improvement in time and avoiding overfitting.

With modifications applied to the model, the model performed slightly differently as seen in the above figure. The validation was slightly higher i.e. **0.5363** in the original experiment shown in (Yoav, 2016) vs. **0.5416** in the modified model due changes applied. However, the new model did not over fit and the best loss value was found within 40 epochs as compared to 250 epochs in the original experiment. The early stopping criteria prevented the network from overfitting and training was performed much faster. For the new model, 150 hidden size produced best loss values as compared to 200 in the original experiment.

As mentioned by the author of original experiment, drop out had huge impact on the training loss and over fitting. Without applying dropout, the networks over fits as seen repeatedly in the grid search. The training time increased considerably with increasing number of neurons and when moving from 1 to 2 hidden layers, which is an expected behaviour.

5.1.1 Generation with Base Model

The generation phase for this experiment remained almost the same as the original experiment. The trained model shown above was fed with a random sequence of samples not seen during the training, to condition the model for 32 time steps (i.e. 2 bars). A sequence of 384 time steps (24 bars) was created by sampling the output probability distribution at each generative step.

Fig. 5.2 shows graph of probabilities with which melodies and harmonies were generated. It could be seen that harmonies were being predicted with a much higher probability than melodies, as highlighted in (Yoav, 2016). Also, listening tests and visual inspection of probability values indicated that harmonies were helping in providing the overall structure to the generated melodies.

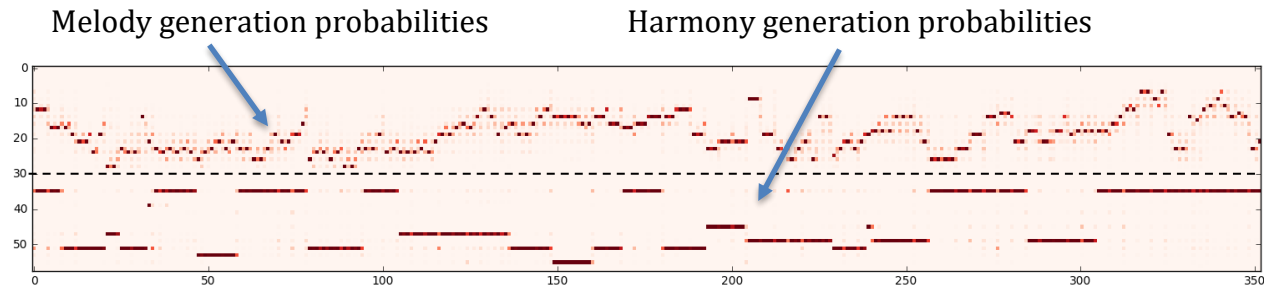


Fig. 5.2 The probabilities for melodic and harmonic components of generated sequences. The saturation of red colour indicates high probabilities. It can be seen that harmonies are being generated with a higher probability than melodies in general. Also, harmonic changes result s melody prediction with a higher probability.

From the piano roll representation of the output, a chord structure can be seen which the network attempts to maintain for most part of the song. The melody was seen as “wandering” within the set melody range and not clearly adhering to a root key. The composition lacked a global with some signs of short term structure which is an expected behaviour of LSTMs.

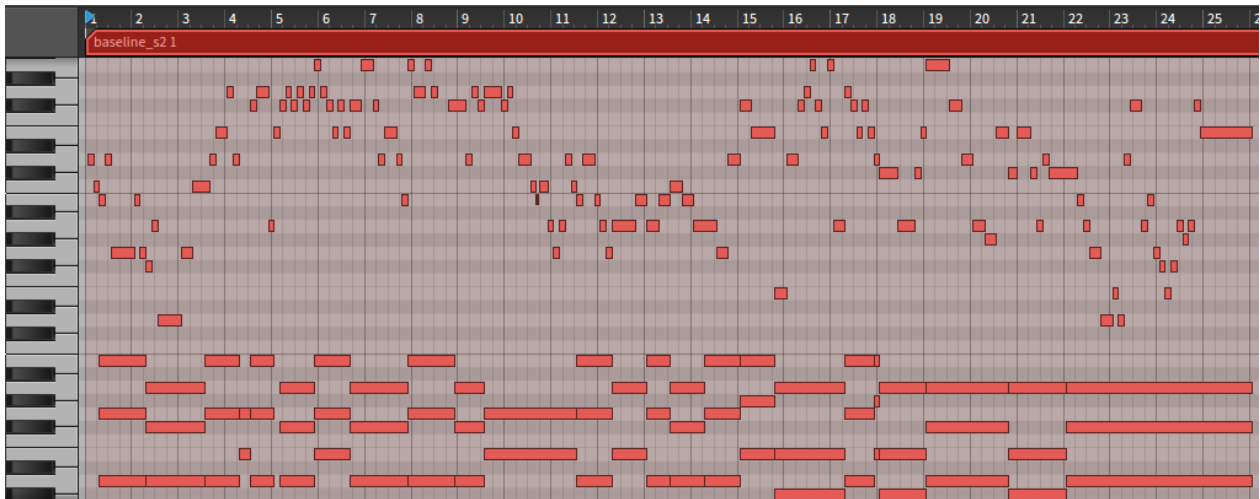


Fig. 5.3 A piano representation of melodies generated by baseline model.
(Baseline Model -S) <https://youtu.be/wbS6HRJ5EHA>

The results of this experiment were found to be close to original experiment in terms of quality of music. This was considered a good starting point for further experimentation and feature augmentation approaches.

5.1.2 Zero-padding Input Data

In order to append length and structure related information to input encoding, the base model was modified to maintain the length of sequences in the dataset as explained in section 4.6. All the sequences were zero padded based on the selected mini-batch length of 128. This allowed preserving the length of the variable length sequences while creating input tensors for TensorFlow. The best model identified in the previous experiment was run again with the new dataset. The model performance under new dataset has been shown in Fig. 5.4.

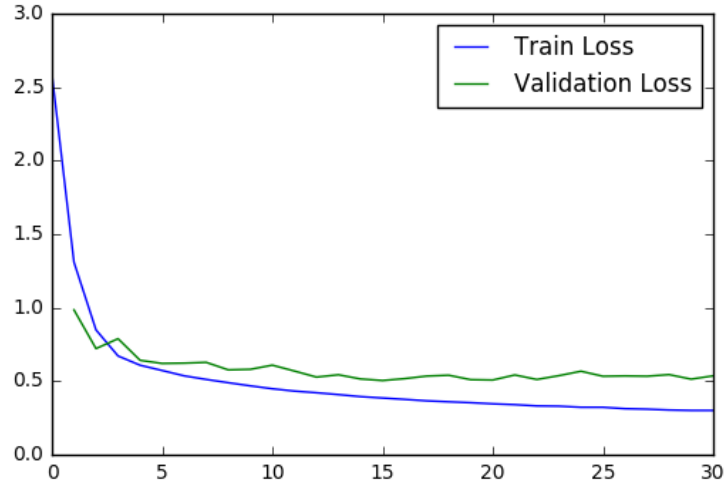


Fig 5.4 Zero-padded showed slightly better performance because of extra inputs comprising of all zeros. (Train loss =0.29601, Valid loss = 0.51209)

The loss slightly improved from the previous model and the new validation loss was found to be **0.51209**. Upon investigation, it was discovered the slight improvement in the loss was due to extra sequences of all zeros. A sequence of all zeros allowed the network to predict another similar sequence with high probability, thus improving the loss calculation. This was not, however, considered a real improvement that might have affected the quality of generated sequences in any way. This also caused mild over fitting compared to previous model in terms of slightly lower training loss, which was **0.29601**, compared to **0.30345** in non-zero padded baseline model. No generation was performed with this model as it was meant to enable structural feature augmentation as shown in following experiments.

5.2 AUMENTING SEQUENCE DURATION COUNTER

A number of different approaches were tried towards augmenting the proposed countdown mechanism as detailed in section 4.5.

5.2.1 Simple Integer Counter

A simple integer countdown was appended to input encoding as described in section 4.5.1. This experiment was performed on the zero-padded dataset and a similar set of grid parameters used in the base model were tried to perform a direct comparison with the base model from first experiment in terms of loss improvement and quality of the generated sequences. This grid search for this experiment is shown in Table 5.2.

The results show an improved validation performance. The best validation was found to be **0.47803**, with two hidden layers each having a layer size of 150, as compared to **0.51209** in the zero-padded baseline model with same parameters but a hidden size of 100. This was an indication that the augmentation allowed the model to learn extra knowledge from engineered features. The training loss **0.53057** was also found to better than zero-padded baseline model which was over fitting with a training loss of **0.29601**. The augmented features and chosen approach prevented the model from overfitting.

No of Layers	Hidden Size	Drop Out	T Loss	V Loss	Ave. Time/Epoch
1	100	1	0.68109	0.58851	33.17
1	100	0.75	0.6655	0.55168	33.14
1	100	0.5	0.74764	0.55167	34.13
1	150	1	0.73704	0.58951	49.77
1	150	0.75	0.66563	0.55985	47.85
1	150	0.5	0.74352	0.55456	46.58
1	200	1	0.77045	0.59109	69.66
1	200	0.75	0.69105	0.56951	67.29
1	200	0.5	0.7122	0.54507	67.38
2	100	1	0.46802	0.51109	70.61
2	100	0.75	0.53439	0.49721	68.34
2	100	0.5	0.64043	0.51089	68.26
2	150	1	0.4414	0.51842	107.59
2	150	0.75	0.62871	0.51507	103.58
2	150	0.5	0.53057	0.47803	97.66
2	200	1	0.51229	0.52209	160.49
2	200	0.75	0.4012	0.49463	144.86
2	200	0.5	0.53011	0.49298	145.69

Table 5.2 Grid search results for simple integer counter augmentation showing improved loss performance.

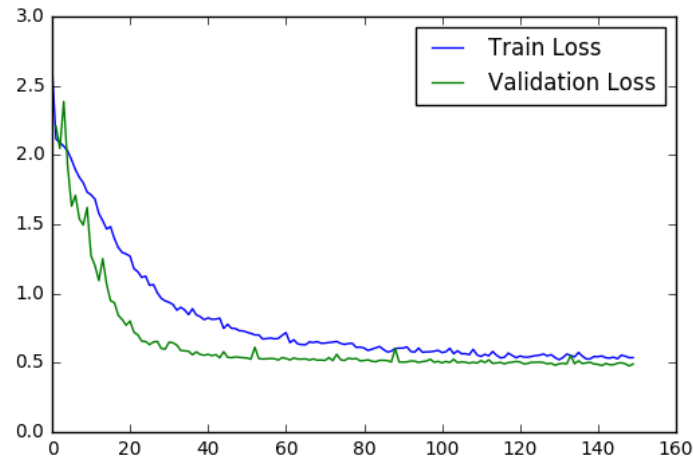


Fig 5.5 Model performance with simple numeric countdown augmentation with first sample having max value = total number of time steps in the sequences, and minimum value = 0 at the last time step. (Training Loss = 0.53057, Valid Loss = 0.47803)

Results from best model in Fig. 5.5 show a huge increase in training time as a result of augmenting the countdown as an integer values. The training time increased from **30** epochs in zero-padded baseline model to **150** epochs with suggested changes. As the MIDI dataset contained sequences of variable length, ranging from 82 to 3588 time steps, large numbers that were fed into the network as counter values heavily impacted the training and it took 150 epochs to achieve a training loss of **0.53057**, as compared to previous experiment where

training loss **0.29601**. The counter led to an improved loss performance but made the model computationally very expensive. This behaviour is shown by model performance in Fig. 5.5.

5.2.2 Findings

Upon investigation, it was realized that although in theory deep neural networks do not necessarily demand normalisation of the input features, there are number of reasons why this might be desirable. Normalising input features speeds training resulting and early convergence. It also prevents the network from getting stuck in a local minimum. Weight decay and other learning techniques usually perform better with normalised inputs. The effect of normalisations is effectively undone by the network as it changes corresponding biases and weights and predicts similar output as with raw values.

5.2.3 Generation with Simple Integer Counter

During the generation phase, a simple feature generator was developed to calculate and iteratively append a countdown vector to generated sequences before feeding them back to the network as shown in section 4.8. A sequence of length 384 (24 bars in 4/4) was generated as shown in the piano roll in Fig. 5.6. A slight improvement in the short-term structure was noticed as compared to the sequence generated by the base model along with the improvement in cross entropy loss. The generated melody sequences showed signs of short term repetitions. Generated sequences also showed slight improvement towards conclusion of the song which was much subtler than seen in the baseline model. This observation was also confirmed by human evaluation. The model still lacked a long-term structure and could not repeat note combinations as structural components.

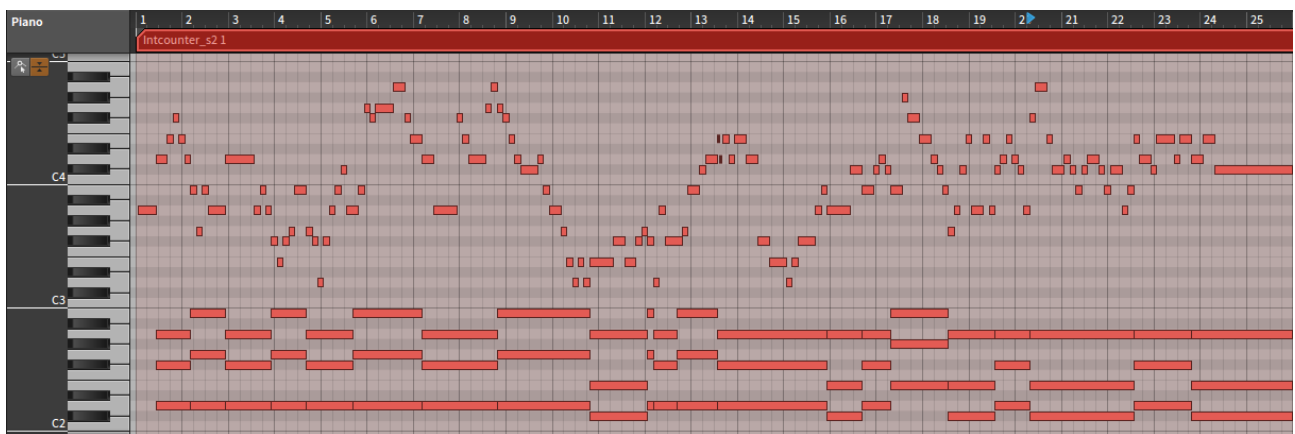


Fig 5.6 Piano roll representation of sequence generated from integer counter augmentation. Some short-term repetitions and a more structured conclusion was seen with this experiment. Simple (Simple Counter –S) <https://youtu.be/BbiMYafkyFk>

5.2.4 Duration Control

In order to further investigate the role of augmented counter values, another experiment was conducted with the same model where the length of counter was kept shorter than total defined sequence length for generation. i.e. the generative process lasted for 384 time steps and the counter was initialized with a value of 256 time steps. The counter reached 0 at 255th time step and stayed 0 for all 128 proceeding time steps. The output generated at this stage is shown in the Fig. 5.7 below.

After 16th bar or 256 time steps, when the counter value reaches zero, the trained model stops predicting further note transitions

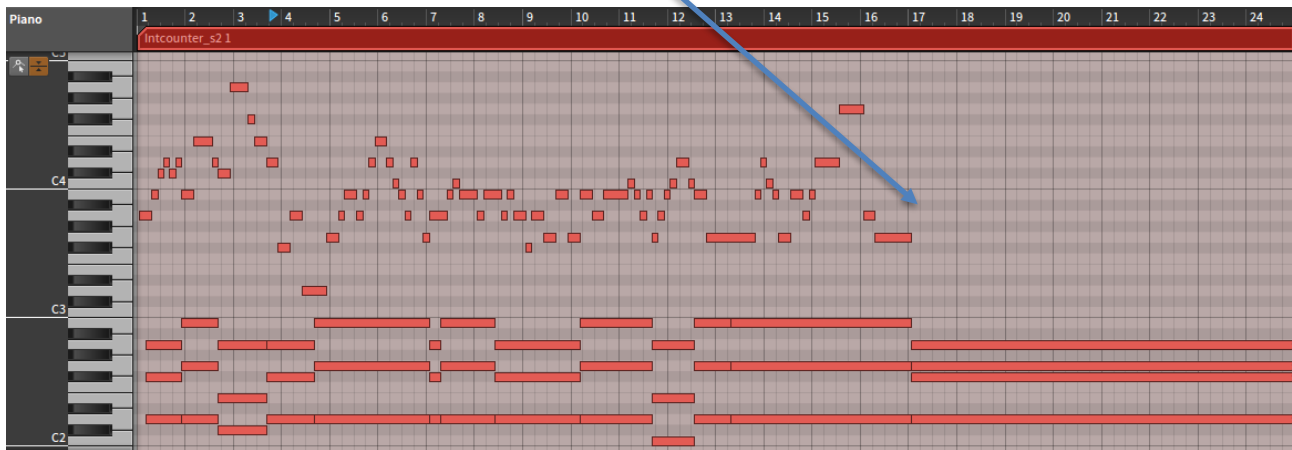


Fig. 5.7 Piano roll representation of generated sequence from counter augmented model showing that the model stops predicting any further note transitions once the counter value reaches zero.

<https://youtu.be/-eFNzipNpEA>

The piano roll shown above indicates that the augmented countdown can influence the generative process and as it stopped the model from generating any note transitions after 16th bar as set by the counter. This result indicate that the counter value does have an impact on the conclusion of a generated sequence. With a number of generations, some conclusions appeared more pleasant with an expected last note, while some sounded off scale and random. However, the network stopped predicting any further melody and harmony transitions after the countdown value reached zero.

The overall results looked positive so a further experiment was tried with same setup while applying further modifications to the counter input.

5.2.5 Modified Counter with Normalization and Shift

Based on the findings from previous experiment, in order to speed up training time and to design the counter on more music-theoretic principles, the countdown values were normalised between 1 (song introduction) to 0 (time step value at the beginning of last note) as explained in section 4.5.1. A further modification of moving the 0 value of the counter back to beginning of last note instead of last time step was also performed as shown. This was with the motivation that implemented modification would allow the network to achieve similar (or better)

performance as compared to previous experiment and the resulting sequence would have some further improvements in the subjective evaluation.

As seen in Table 5.3, the best validation loss value of **0.47029** (shown in red), very similar to **0.47803** as in the previous experiment was achieved with 2 hidden layers of size 200 and drop out of 0.5. This model also showed some signs of overfitting with training loss at **0.27951**. A similar validation loss value of **0.47357** (shown in blue) was also seen in a slightly simpler model with 100-layer size, showing an improved training error of **0.39425**. This model was chosen for discussion and generation.

No of Layers	Hidden Size	Drop Out	T Loss	V Loss	Ave. Time/Epoch
1	100	1	0.32896	0.55106	31.93
1	100	0.75	0.32045	0.51925	31.5
1	100	0.5	0.39416	0.51017	32.52
1	150	1	0.23093	0.54249	38.46
1	150	0.75	0.27439	0.49835	40.58
1	150	0.5	0.32207	0.49207	38.16
1	200	1	0.18841	0.53576	51.42
1	200	0.75	0.22439	0.51228	50.99
1	200	0.5	0.25068	0.48468	50.06
2	100	1	0.2154	0.57109	59.96
2	100	0.75	0.31165	0.48653	57.53
2	100	0.5	0.39425	0.47357	58.57
2	150	1	0.18107	0.55239	87.31
2	150	0.75	0.25594	0.48468	81.31
2	150	0.5	0.3108	0.47183	79.4
2	200	1	0.1445	0.57807	108.12
2	200	0.75	0.20278	0.49504	108.43
2	200	0.5	0.27951	0.47029	108.32

Table 5.3 This table show the output of grid search applied to the model trained on dataset containing normalized and shifted duration countdown augmentation. The model loss is shown in red, with a more suitable model having less over fitting shown in blue.

The selected model (highlighted in blue) shown in Fig. 5.8, trained much faster i.e. **55** epochs as compared to **150** epochs in previous experiment using non-normalised counter, thus justifying the need for input normalisation as explained in section 4.5.2. The model was slightly overfitting with the training loss **0.39425**, compared to **0.53057** in previous experiment. This was also considered an impact of normalisation as large integer counter values were hard to learn with sequences of different length. The normalised values allowed the model to train efficiently as similar values were seen with all sequences.

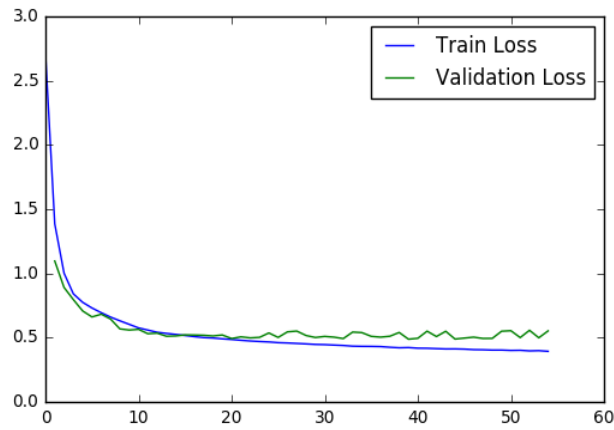


Fig. 5.8 Best model identified with normalised and shifted countdown augmentation. The model shows a training loss of 0.39425 and validation loss of 0.47357

5.2.6 Generation with Modified Counter

Using the same model for training and generation as used in previous experiment, the counter values were normalized and augmented to input encoding. Following piano roll representation shows the output of generation process.

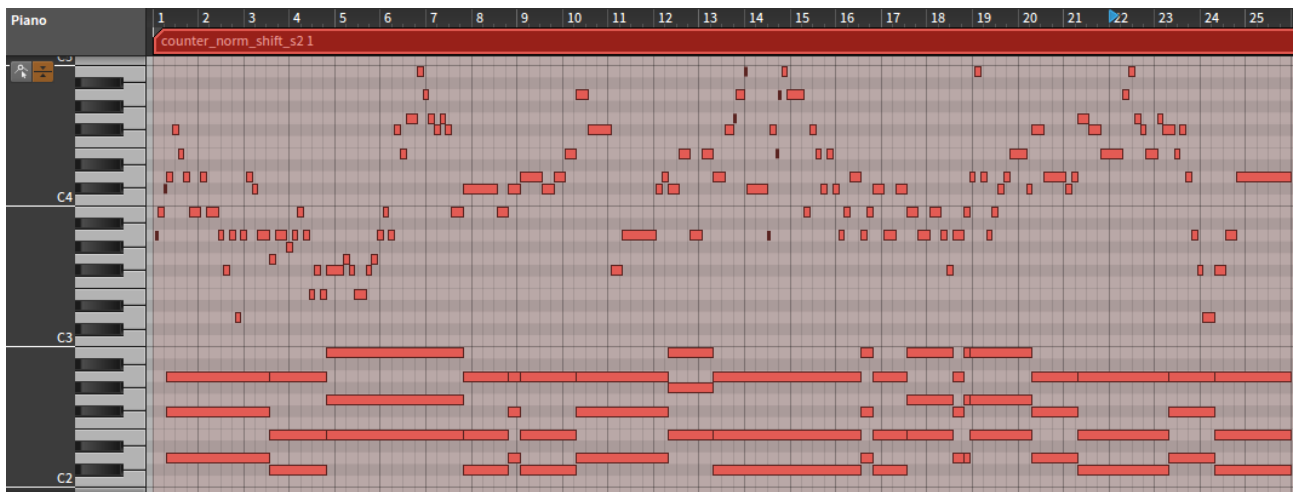


Fig. 5.9 Piano roll representation of generated sequences from normalized and shifted countdown augmentation show similar performance as the integer values and offers similar listening experience.
(Modified Counter -R) <https://youtu.be/CQOCM-SgzNs>

The results in Fig 5.9 visually appeared and sounded very similar to initial experiment with non-normalised counter. Some short term structural repetitions and a decent conclusion was observed as in previous case. The effect of shifting the counter back could not be clearly calculated as a number of sequences were generated by alternating shift flag in the code and no clear difference was observed. However due to its theoretical justification towards song conclusion as mentioned in section 4.5.2, it was kept as a part of input encoding for further experiments.

Duration control experiment, as performed during previous stage were also tried resulting as similar output i.e. the network stops predicting further transitions once the counter reaches zero. Visualising the probabilities of sequences with counter reaching zero before end of the sequence shows a rest value for melody and no further harmonic transitions, as shown in the following figure.

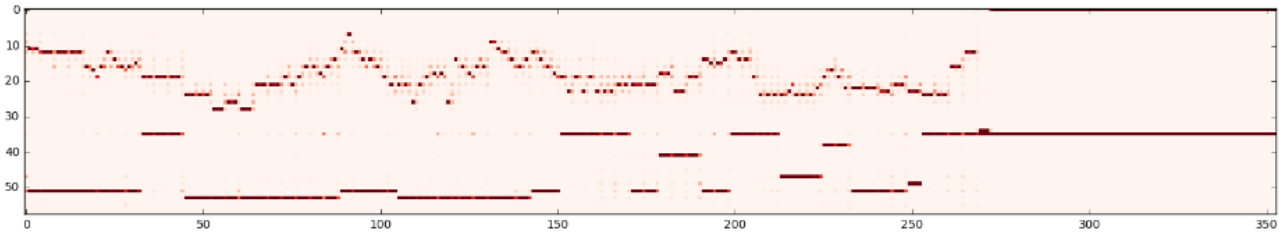


Fig. 5.10 Visualising probabilities with which melodies and harmonies are generated during the generation phase highlight that no further transitions are predicted once the counter value reaches zero, as observed with integer values.

5.3 AUGMENTING METRICAL INFORMATION

The positive results seen in duration feature augmentation provided ground and motivation for continuing with the structure augmentation experiment and further investigations were performed for possible candidate features. For the next experiment, metrical features were further augmented with the input encoding from previous experiment as explained in section 4.5.3. Initially two one hot vectors counting counter beats and whole bars were planned for this experiment (as shown in Fig. 4.10), but this resulted as very poor model performance and very low quality generated sequence. Considering LSTM's capability to remember and generate short term temporal associations, the quarter beat counter was removed from input encoding. Metrical feature augmentation experiments were performed with a 4 bit one-hot vector where each bit pointed towards a quarter beat with 4/4 assumption.

5.3.1 Augmenting Countdown and Metrical Information to Complete Dataset

The network was trained with normalised and shifted countdown values from previous experiment, and a one-hot vector of length 4 to count the bar as mentioned above. A grid search was conducted while lowering the values of drop-out with parameter values as shown below. The drop out range was further lowered as it seemed to be helping the models achieve better performance.

Drop Out: [0.75, 0.5, 0.3]

Number of Hidden Layers: [1, 2]

Hidden Layer Size: [100, 150, 200]

No of Layers	Hidden Size	Drop Out	Train Loss	Valid Loss	Ave. Time/Epoch
1	100	0.7	0.3565	0.5029	33.083
1	100	0.5	0.4047	0.5052	31.62
1	100	0.3	0.4302	0.4982	33.35
1	150	0.7	0.2517	0.508	42.86
1	150	0.5	0.2922	0.4894	43.32
1	150	0.3	0.394	0.5115	42.41
1	200	0.7	0.2117	0.5032	55.80
1	200	0.5	0.245	0.4915	52.84
1	200	0.3	0.2901	0.4919	51.82
2	100	0.7	0.3026	0.4872	61.54
2	100	0.5	0.3669	0.482	62.92
2	100	0.3	0.5204	0.4998	63.42
2	150	0.7	0.2668	0.4887	84.48
2	150	0.5	0.2934	0.4806	87.59
2	150	0.3	0.4295	0.4855	82.88
2	200	0.7	0.2161	0.4933	115.34
2	200	0.5	0.249	0.4723	115.81
2	200	0.3	0.3476	0.4693	116.55

Table 5.4 Showing the impact of 4/4 metrical encoding to complete dataset, a slightly lower loss value than the one from previous experiment can be seen in the last run.

It was noticed that the validation loss came down from previous experiment, indicating that the network learnt some extra information about training data. However, the training time with 2 layers of size 200 went up to **116.5593** sec/epoch from **58.57** sec/epoch with counter augmentation. Also, a lowered drop out value helped the model achieve this performance, but at a much higher computational cost. Due to the dataset having a number of different time signatures associated to sequences, while the metrical encoding only catered for 4/4 sequences, the aesthetic quality of generated sequences was not found to be very pleasant. The metre of generated sequences appeared to shift within different time signatures as a result of appended features resulting as unexpected changes in metre and scale. This was further confirmed during listening tests. This also confirms the view that a high predictive performance of a model may not necessarily relate to the quality of music it generates. The conclusion of the sequence, however was very similar to previous experiments and did not sound totally random. The best model from the grid search is shown below in figure 5.11.

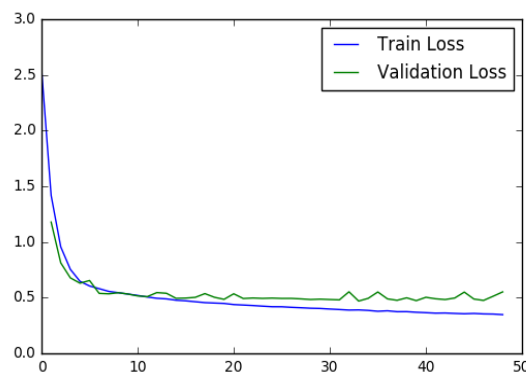


Fig. 5.11 Best model identified with countdown and metrical information augmentation on complete Nottingham dataset. The model shows a training loss of 0.3476 and 0.4693

A generated sequence from this experiment is shown in Fig. 5.12. This sequence along with a randomly generated sequence from the model were used for listening tests.

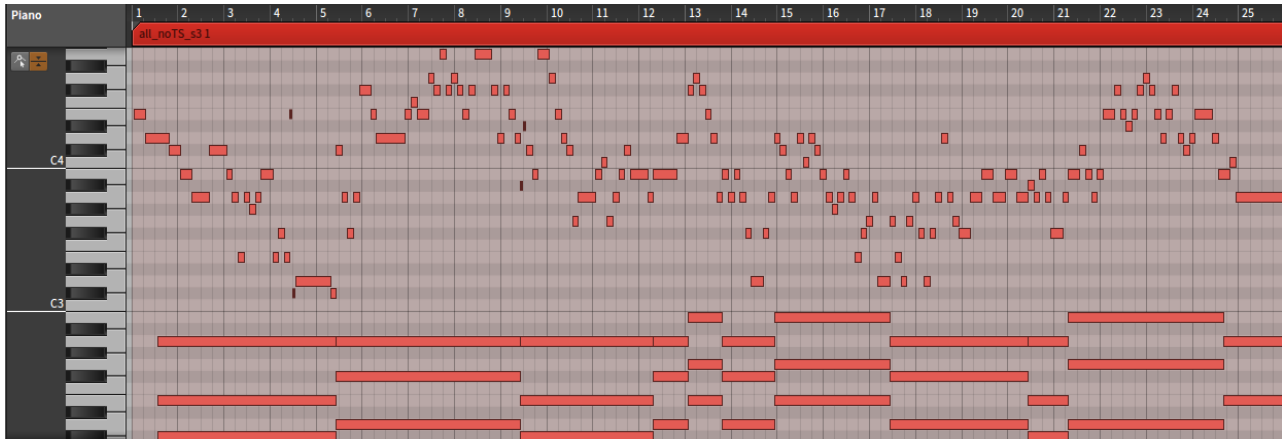


Fig. 5.12 Piano roll representation of generated sequences from duration and metrical information augmentation show good predictive performance yet offers a poor listening experience.

(Nottingham All Feat -R) https://youtu.be/1_ooA3Ailpg

5.3.2 Augmenting Metrical Information to 4/4 Sequences

In order to deal with the observation made above, it was decided to extract all 4/4 sequences as a subset of corpus to try the metrical augmentation approach which was catered for these sequences. Also, as it was obvious from previous experiments that a drop out value of 1 i.e. no dropout was not helpful. A reduced grid search was performed with following parameters values.

Drop Out: [0.5, 0.3]

Number of Hidden Layers: [2]

Hidden Layer Size: [100, 150, 200]

Table 5.5 shows a similar validation loss as experienced in the previous experiment. i.e. **0.46286** with 4/4 music only, as compared to **0.4693** for complete dataset in previous experiment. The training, however, completed in much shorter time i.e. average **85.96** sec/epoch (4/4 music) vs. **116.55** sec/epoch (complete dataset). The reduction in training time was due to reduced data size as 4/4 sequences roughly make up about 50 percent of Nottingham dataset. Also, a reduced training loss was an indication that the model maybe slightly overfitting due to reduction in data size and sequences being very similar to each other in metre and carrying metrical information.

No of Layers	Hidden Size	Drop Out	Train Loss	Valid Loss	Ave. Time/Epoch
2	100	0.5	0.29408	0.47204	52.08
2	100	0.3	0.49723	0.48458	52.93
2	150	0.5	0.28959	0.47337	68.68
2	150	0.3	0.35095	0.47032	70.70
2	200	0.5	0.27501	0.46910	85.17
2	200	0.3	0.30637	0.47507	87.14

Table 5.5 Grid search performed using a reduced parameter range with metrical encoding using 4/4 sequences from the dataset.

Fig. 5.13 shows the performance of the best model found in the grid search. The model clearly shows lower training loss of **0.27501** than previous experiment, due to reduced number of examples seen during training. This is considered a common behaviour with neural networks.

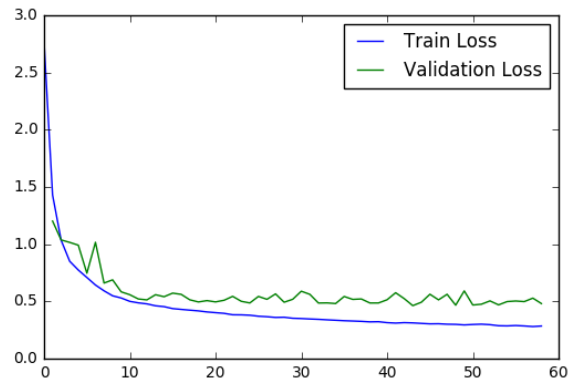


Fig. 5.13 Best model identified with countdown and metrical feature augmentation with 4/4 music sequences. The model shows a training loss of 0.28484 and valid loss 0.46286.

5.3.3 Generation with Metrical Feature Augmentation

A number of generations were performed with the trained model as described above. The results of generation process were found to be very interesting as shown in the examples below. A number of melody rises and falls (melodic motions) can be seen from the piano roll representation, however, unlike previous models, the network learnt to maintain the short-term structure more effectively as it attempted to maintain the scale and rhythm of generated sequences and also performed very well in terms of maintaining the metre of sequences in 4/4. Improvement in the short-term structure seen here is mainly due to the model training on 4/4 sequences only and for each sequence, extra information about placement of melody and harmony notes within bars was augmented to the network. An example of this behaviour can be seen in the piano roll representation of a generated melody from this model. The network tries to maintain 4/4 metre in most part of the song and learns to change the chord on a strong beat as shown in Fig. 5.14.

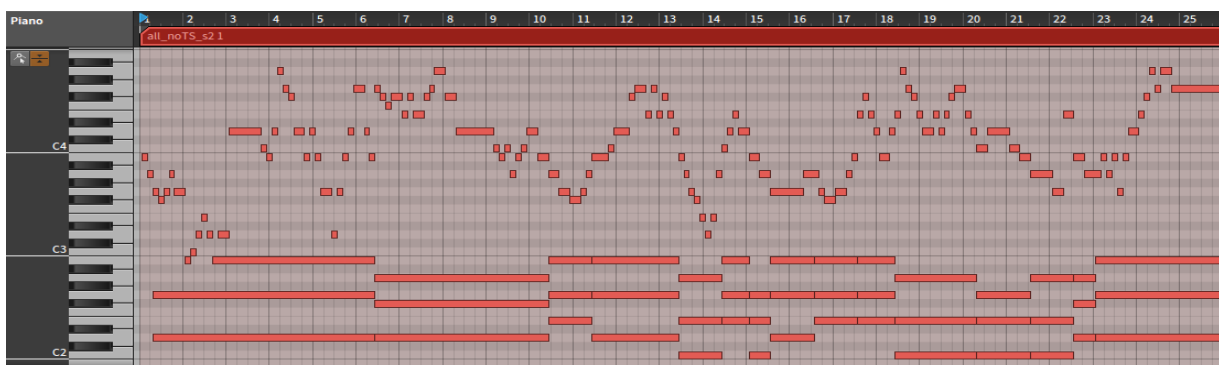


Fig. 5.14 Metrical and duration related engineered features help the model in generating more structured sequences with higher number of short term repetitions while returning to root note and maintaining the rhythm and metre of the song for most part.

(4/4Data All Feat -S) <https://youtu.be/yeLiBsUsNA>

In another generative example as shown below in Fig. 5.15, the network changed the rhythm and style in the middle of the generative process. The network did not make an attempt to resume the original rhythm highlighting that the model still lacks in learning and producing long term musical structure. It was considered to be an effect of mini-batching as the network was trained on mini-batches 128 time steps and can easily learn the association of melodies and harmonies within that range.

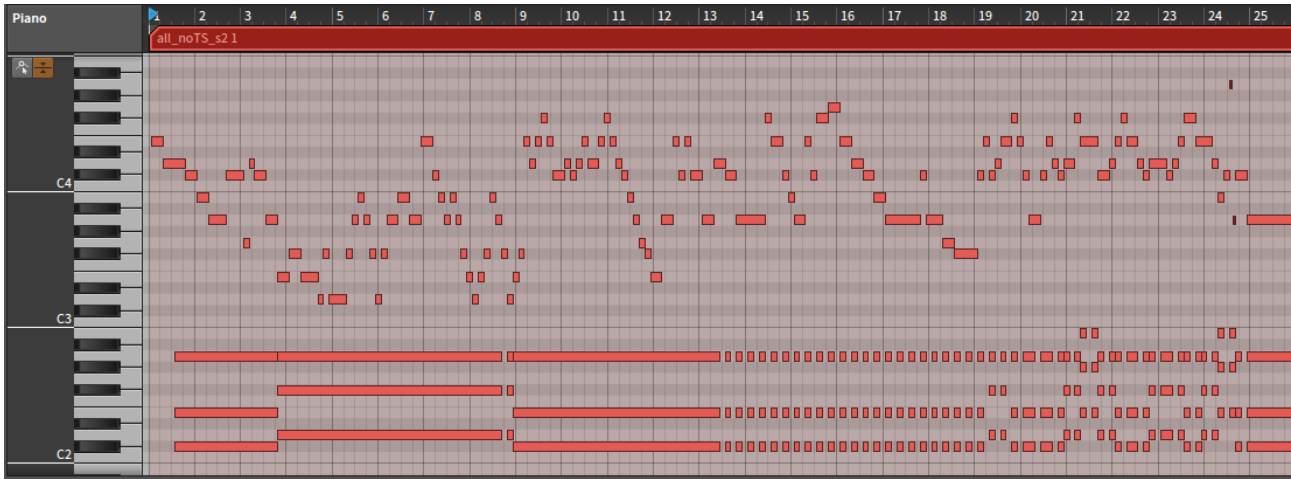


Fig. 5.15 In this generation experiment, the model changed the rhythm and style of generated sequences half way through generative process and did not attempt to resume the original style.

(4/4Data All Feat -R) <https://youtu.be/IQtG72ymfS8>

The experimentation at this stage was concluded with an objective understanding of the fact that duration and metrical features do allow RNN models to learn extra information about the sequences. Further augmentation with more labelled training datasets annotated with global structure information e.g. chorus, verse and bridge etc. can also prove to be beneficial towards allowing the model to “return” to an original scale and style. However, no such datasets suitable for deep learning domain exist at present.

5.4 MODEL COMPARISON

Based on the experiments above, a model comparison showing loss performances of all models is shown in Table 5.6. Baseline validation loss of **0.54160** improved with counter augmentation, to **0.47357**. This is partly due to zero padding as mentioned earlier which brought the loss down to **0.51209**. Counter augmentation clearly improves the loss further. The final StructLSTM features augment loss value is further improved to **0.46910**. The model, however shows sign of overfitting with the training loss of **0.27501** as compared with earlier training loss values. This is mainly due to model training on roughly 50% of the data having 4/4 time signature, as shown earlier in Fig. 4.3. Therefore, the models cannot be directly compared. A larger data set with more training example can help overcome this problem. Early stopping criterion applied allowed to model to overfit any further. A drop out value of 0.5, or lower with complete dataset can be seen. Without these measures the model could overfit further.

	Number of Layers	Hidden Size	Drop Out	Train Loss	Valid Loss	Ave. Time/epoch
Baseline	2	150	0.5	0.30345	0.54160	74.48
Simple Counter	2	150	0.5	0.53057	0.47803	97.66
Modified Counter	2	100	0.5	0.39425	0.47357	58.57
4/4 features (All data)	2	200	0.3	0.3476	0.4693	116.55
StructLSTM 4/4 features (4/4 data)	2	200	0.5	0.27501	0.46910	85.17

Table 5.6 A comparison of five experiments shown earlier within this section. StructLSTM based model shows best loss value, but with some overfitting.

All the models perform best with 2 hidden layers and a hidden size of 150 – 200. The training time increased with augmentation of simple integer based counter due to use of large values which came down to **58.57**, the lowest for all experiments.

These observations clearly show that augmented features had an impact on training and generation phases. The loss values do not reflect the aesthetic appeal in generated sequences, as seen by augmenting 4/4 metrical feature to whole dataset. All models generated sequences which could be identified with their aesthetic attributes. Due to a lower validation loss, and partly due to slight overfitting, the StructLSTM generated sequences showed more aesthetic quality than all other experiments. During listening tests, the StructLSTM generated sequence's score was only beaten by a human performance. This highlights the impact of structure augmentation on musical data. A deeper investigation with a bigger dataset may lead to further conclusions. A subjective analysis of generated sequences is provided in next chapter.

The sequences generated in above experiments were selected for subjective evaluation, along with other randomly generated sequences from same models. The listening tests details are provided in next chapter.

A zip file containing feature augmented datasets created in this section is being attached as digital submission in Appendix D.1 (**Appendix_D.1_datasets.zip**).

A zip file containing models trained with feature augmented datasets is being attached as digital submission in Appendix D.2 (**Appendix_D.2_trained_models.zip**)

Also available at the StructLSTM GitHub repo
(<https://github.com/ShakeelRaja/structlstm>)

CHAPTER 6. SUBJECTIVE EVALUATION AND RESULTS

A musical composition holds its true meaning through subjective understanding and interpretation of the listener. The evaluation of beauty and aesthetics, found in music is often defined as a subjective experience (Briot & Pachet, 2017). An objective evaluation showing an improvement is the loss function of a neural model may not necessarily guarantee an output that will be appreciated more by the listener as model's loss minimization does not cater for subjective interpretation of generated sequences. The numeric and relational understanding of the input musical sequences by a neural network does not relate directly to cognitive process used in musical understanding. Evaluating algorithmic music is hence considered to be highly challenging as compared to other domains including NLP and computer vision etc.

6.1 EVALUATION FRAMEWORK

As the main objective of this research was identified as an attempt to improve the structure of generated composition, a simple subjective evaluation of the generated compositions was planned to measure the aesthetic and subjective aspect of generated music. The inspiration for this approach comes from Alan Turing's *Turing Test* which is now heavily used in the field of AI to measure whether a machine can demonstrate a level of intelligence, similar to that of human beings (Hiraga et al., 2004). This experiment was not aimed at developing algorithms that equate or outperform human compositional skills, but to measure the impact of feature augmentation against a base model. A slight variation to the classical "either-or" style of Turing test was used by introducing linear scales with a range of 1 to 5 to measure the level of improvement. Some elements for developing the listening test were taken from the framework developed by Pearce and Wiggins, titled "Towards A Framework for Evaluating Machine Compositions" (Pearce & Wiggins, 2001). The framework developed for this evaluation comprised following key points.

1. Identifying the compositional aims.
2. Inducing compositions from base model and humans to allow listeners to compare against, and detect any change in aesthetic quality of music.
3. Identify candidate compositions from different stages of experimentation
4. Engaging listeners and evaluating listeners' feedback against set objectives and compare with objective evaluation.
5. Ensuring All ethical requirements are met towards user information, participation and consent.

6.1.1 Identify Compositional Aims

The compositional aims i.e. learning from augmented structural knowledge and generating better structure has been described and addressed in detail throughout the report. To measure the impact of duration and metrical feature augmentation to the input encoding at different stages, following questions were asked, based on expected improvements in the generated sequences.

- Q1. In your opinion, what is the likelihood that the music is composed by human vs. machine?
Q2. How do you rate the long-term structure of this composition?
Q3. How do you rate the short-term structure of this composition?
Q4. In your opinion, how well does the composition come to a conclusion (ending)?
Q5. How would you score overall quality of this musical sequence?

6.1.2 Induce Human and Base Model Compositions

In order to provide grounds for comparison, two sequences generated by the base model were used where the first sequence was selected by the author out of many generated melodies, and a second sequence was randomly generated and included to show model's random generation performance. Similarly, two sequences were randomly drawn from the Nottingham dataset to serve as human compositions.

6.1.3 Identify Candidate Compositions

For evaluation, two sequences were selected from each key stage within the experiment on the criteria used with base model above i.e. one selected and one randomly drawn sequence.

Following three key stages were identified for the experiments:

1. Countdown augmentation with the normalised shifted countdown.
2. Countdown and metrical information augmentation on complete dataset.
3. Countdown and metrical information augmentation on 4/4 pieces from the dataset.

A total of 10 sequences, based on the criteria above, were presented to users in an online listening and scoring setup. The sequences were randomly shuffled regularly to avoid bias towards scoring as a result of order to these sequences.

6.1.4 Collect Listeners' Feedback

In order to conduct the listening tests within a tight timeframe, an online questionnaire was developed using Google Forms (Section 6.2). The sequences were converted into video files and uploaded on YouTube for easy linking with Google form. A number of listeners from different musical backgrounds were invited to evaluate the sequences. Ethical requirements during this test were ensured as required by the University, and are provided in **Appendix E**. Listeners were requested to read a **participant information sheet (Appendix E.1)** and sign a **consent form (Appendix E.2)**. Users were also shown a document as an **introduction** to questions and how to score the sequences (**Appendix E.3**). For classifying responses, users were requested to specify their musical experience from following:

- Group 1: Not interested in music
- Group 2: Casual listener, appreciate good music but not familiar with musical concepts
- Group 3: Understand music theory and/or sing/play a musical instrument
- Group 4: Music professional / enthusiast

This initial information was followed by an introduction to the project scope, reason for the survey and help on answering questions. A total of 10 sequences were presented to the listeners in the format given below. For each sequence, users were required to give a score to quality, aesthetics and structure of sequence as shown in Fig. 6.1.

6.2 ON-LINE LISTENING TEST

A number of musical sequences generated with models identified in previous section, were used for subjective evaluation through Turing style listening tests. The online test presented to the users can be viewed at:

Structure Augmented Long Short-Term Memory Network (StructLSTM) for Music Composition

<https://goo.gl/J24GVq>

Listening Test 1

Listen to the music based on the criteria described above.

b11 Structure Augmented LSTM
by SHAKEL BAKR
Machine (generated by LSTM) - Center City, London, UK
CITY
*Please rate the music based on these criteria. Your opinion will be highly appreciated.

In your opinion, what is the likelihood of the music composed by human vs. machine ? *

1 2 3 4 5

Human ☐ ☐ ☐ ☐ ☐ Machine

How do you rate the long term structure of this composition? *

1 2 3 4 5

Excellent ☐ ☐ ☐ ☐ ☐ Poor

How do you rate the short term structure of this composition? *

1 2 3 4 5

Excellent ☐ ☐ ☐ ☐ ☐ Poor

In your opinion, how well does the composition come to a conclusion (ending) ? *

1 2 3 4 5

Excellent ☐ ☐ ☐ ☐ ☐ Poor

How would you score overall quality of this musical sequence?

1 2 3 4 5

Excellent ☐ ☐ ☐ ☐ ☐ Poor

BACK **NEXT** Page 3 of 12

Fig. 6.1 An example screenshot of a listening test presented to the listeners. All listeners were requested to undertake 10 of such tests

Musical sequences presented to the listeners with their YouTube links are listed below. As the output of the network is totally probabilistic, the sequences it generated varied in quality. *Selected sequences* below are chosen from a set of four generated sequences based on aesthetics and structure. *Random sequences* were randomly generated to show the total output of the network without any further selection to avoid possible selection bias. For each model, one random and one selected sequence was used in the test. These sequences were presented to listeners in a random fashion during the test. For selected sequences, a YouTube link and representative name (**in brackets**) for representation in graphs and later discussion is given below¹.

Human Composition 1 from Nottingham Dataset (**Human 1**)

<https://youtu.be/7PRf8Ulb5B4>

Human Composition 2 from Nottingham Dataset (**Human 2**)

<https://youtu.be/41k3YWB0Uys>

Baseline model – Selected sequence (**Baseline Model –S**)

<https://youtu.be/wbS6HRJ5EHA>

Base Model – Random Sequence (**Baseline Model –R**)

<https://youtu.be/R995PCMKySk>

Simple Integer Duration Countdown Augmentation – Selected Sequence (**Simple Counter –S**)

<https://youtu.be/BbiMYafkyFk>

Normalized and Shifted Duration Countdown – Random Sequence (**Modified Counter –R**)

<https://youtu.be/CQOCM-SgzNs>

Complete Dataset with Durational and Metrical Counters – Selected Sequence
(**Nottingham All Feat –S**)

<https://youtu.be/kXfyWCXls2I>

Complete Dataset with Durational and Metrical Counters – Random Sequence
(**Nottingham All Feat –R**)

https://youtu.be/1_ooA3Ailpg

Subset of Data, 4/4 sequences with Duration and Metrical counters – Selected Sequence
(**4/4Data All Feat –S**)

<https://youtu.be/yeLiBsUsNA>

Subset of Data, 4/4 sequences with Duration and Metrical counters – Random Sequence
(**4/4Data All Feat –R**)

<https://youtu.be/IQtG72ymfS8>

In order to evaluate the performance of sequences against given compositional aims, the data obtained from Google forms was summarized to evaluate overall feedback, and feedback from specific listener groups. A total of 40 responses were received for this test which ran for over a week.

¹ Above samples can also be downloaded from
https://github.com/ShakeelRaja/structlstm/tree/master/audio_samples

6.3 SUBJECTIVE EVALUATION OF GENERATED SEQUENCES

Following section highlights visual analysis of user responses and brief evaluation and discussion around findings.

6.3.1 Participant Groups

The pie chart shown in Fig 6.2 shows listeners' participation based on their background.

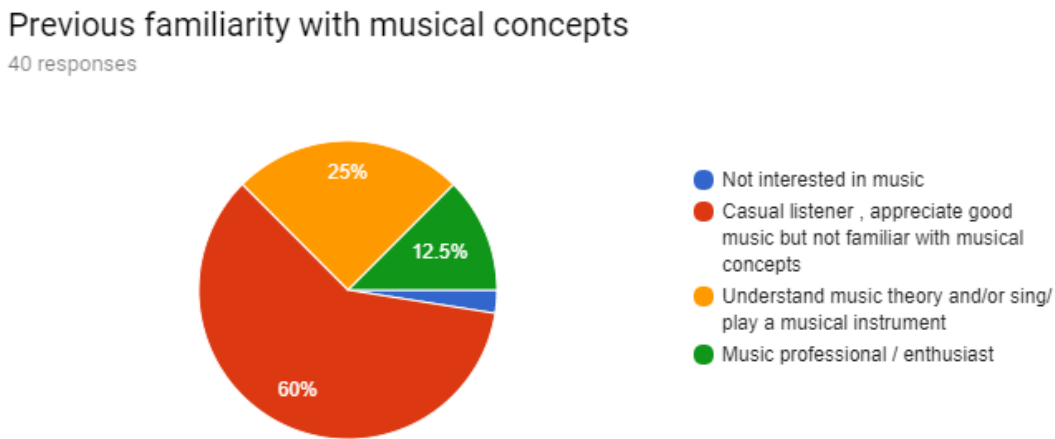


Fig 6.2 A pie chart showing user participation based on their musical knowledge.

The pie chart above highlights the fact that most participants in the survey were casual listeners (24 users making up 60% of all participants). Some listeners from this group also mentioned that they were not familiar with music structure related terminologies used in the description and questions. These listeners were provided extra information and examples to help them understand basic ideas around a musical composition's long term and short-term structure. 10 listeners (25%) belonged to the group who understood music theory or played a musical instrument. 5 listeners were professional musicians/enthusiasts making up 12.5% of listeners and 1 of the listeners had no interest in music (2.5%).

6.3.2 Average Scores

Fig. 6.3 shows the average score for all the included sequences and contribution of each user group towards the score. This average was calculated in two steps as given below:

- For each question in a sequence, the question's average score was calculated as average of individual scores given by all the listeners.
- Each sequence was scored using the average question scores (from last step) in each sequence and taking an average over all question scores related to that sequence i.e. $\text{Avg}(Q1, Q2, Q3, Q4, Q5)$.

It must be highlighted that these scores are based on a scale where 1 represents "excellent" and 5 represents "poor", similarly 1 represents "human composition" and 5 represents "computer

generated music”. Hence a score closer to 1 may appear low but that is desirable for the models. Same convention is used in later graphs.

Results show that music professionals/enthusiasts and listeners who understand music theory or play an instrument can easily differentiate between computer generated music and human compositions. Casual listeners who were not well versed with musical structural knowledge, or those not interested in music found it hard to differentiate between the two. The representative coloured lines for each user group clearly reflect this fact with red and orange lines (groups with musical knowledge) giving scores towards extreme. Green line (casual listeners) are indecisive in general as it appears more around the mean value of 3. It

Average Listener Scores and Group Contribution

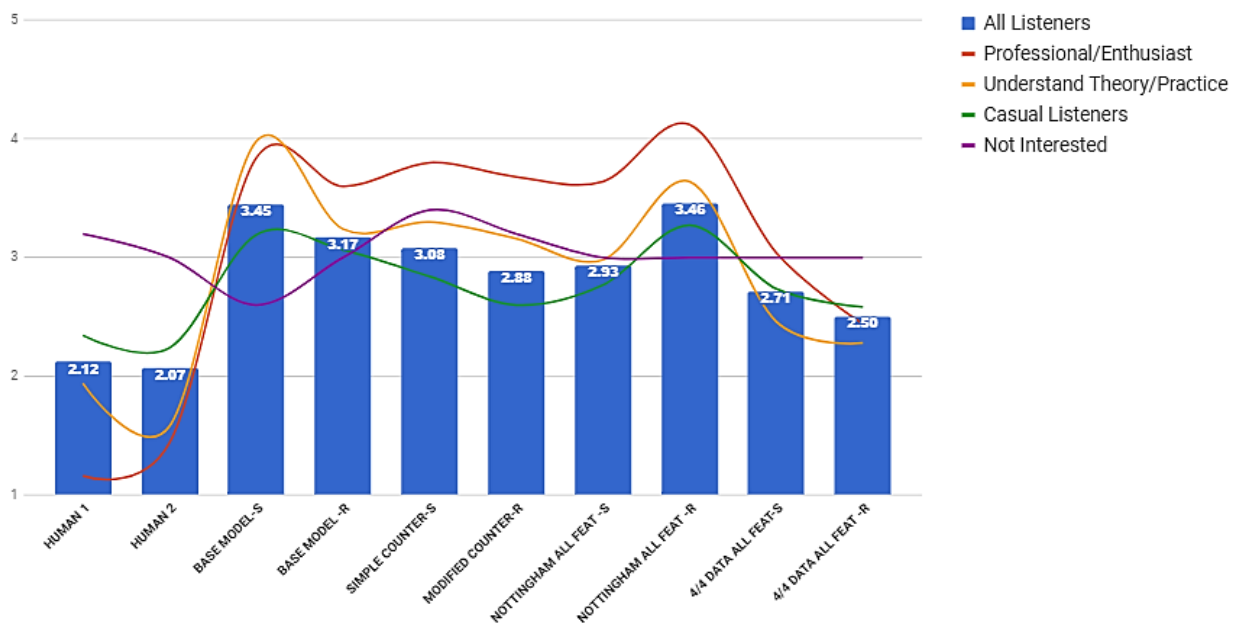


Fig 6.3 Average user scores for all sequences with contribution from each listener group. The feature augmentation experiments are seen to be performing better than baseline models in general, except where metrical information with 4/4 assumption is applied to whole dataset (Nottingham All Feat -R), showing a low score. (Note: 1 reflects excellent and 5 is reflects a poor performance)

Looking at average scores, it is evident that countdown augmentation gives better results compared to the base model. The randomly selected sequence by modified counter (Modified Counter -R) scored **2.88** vs randomly selected base model (Base Model -R) with a score of **3.17**. This improvement is primarily a result of a better song conclusion, a slightly improved short-term structure and an improved general quality as noticed during objective evaluation and confirmed by listeners’ responses. Augmenting 4/4 metrical features to the whole dataset led to generation of melodies that were not pleasant and hence did not perform very well. Randomly generated sequence with all features on whole dataset (Nottingham all Feat -R) performed worst in the test with a score of **3.46**. Lower performance in these sequences is due to an attempt to train music of different structure on 4/4 structural features. Augmenting 4/4 metrical features and a duration countdown to 4/4 selected sequences from data corpus led to generation of aesthetically pleasing sequences as seen in Fig 5.13. The best performing model was the randomly generated sequence with all features on 4/4 dataset (4/4 Data All Feat -R)

with a score of **2.5**. Human composition (Human 2) was given the best score of **2.07**. The area where generated melodies performed poor in general was towards their long-term structure as indicated by listeners' scores.

It was also interesting to see that listeners preferred randomly generated sequences over hand-picked ones. To further investigate the reasons behind these scores and to evaluate listeners' feedback on the questions related to structural quality of the sequences, four random sequences (**Human 2, baseline Model -R, Modified Counter -R, and 4/4 Data All Feat -R**, as shown above) were selected along with the feedback from listeners with a musical background (Groups 3 and 4). Random sequences in each case were selected to avoid any selection bias. This decision was taken to benefit from the musical knowledge that these groups possessed and to evaluate their feedback for individual questions for randomly generated sequences. It did, however, reduced the number of total listeners from 40 to 15. The average scores for each question in each sequence in following cases is based on individual scores only by Groups 3 and 4 and may not directly reflect the averages shown in Fig. 6.3.

6.3.3 Evaluating Human Compositions

Human composition (Human 2) was investigated first revealing information provided in Fig. 6.4.

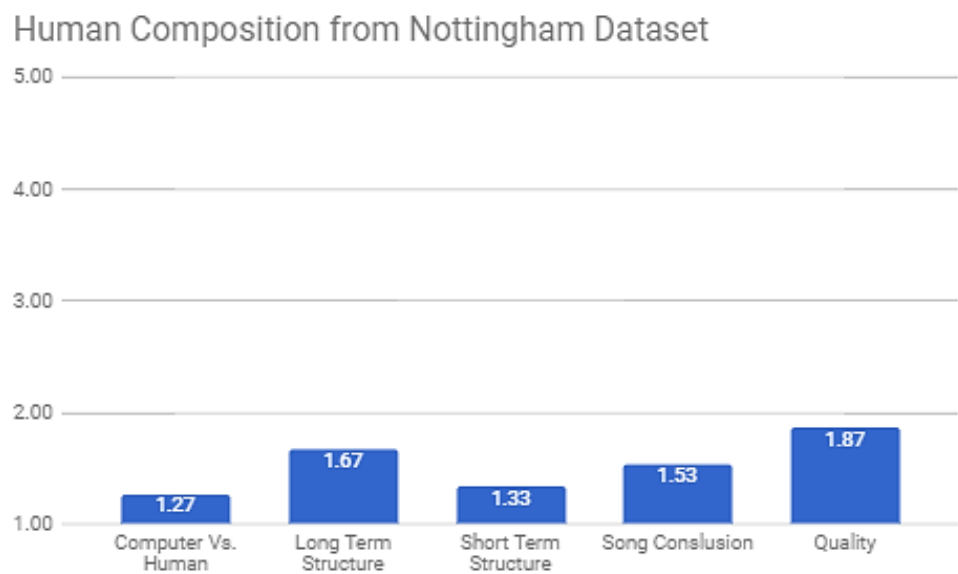


Fig 6.4 Subjective evaluation of human composition by listeners having musical background (Groups 3 and 4)

In Fig. 6.4, it can be seen that users who had a musical background could easily identify the long term and short-term structure found within human compositions and on average gave it a high score closer to 1, identifying it as a sequence with good aesthetic quality having a defined structure and a pleasant conclusion. The reason for a slightly poor score in quality **1.87**, and long term structure **1.67** can be attributed to listeners' preferences towards music genres.

6.3.4 Evaluating Baseline Model Generation

The average feedback from same groups of listeners was evaluated for randomly generated sequence from the baseline model (Baseline Model –R) as shown in Fig. 6.5. This model did not perform very well and was easily identified as being very close to computer generated music with an average score of **4.53** from listeners with musical background. The sequence got a poor score in other questions and overall response was below average. The sequence lacked human-like qualities as seen with the human composition previously. The sequence ended abruptly making it sound even less aesthetically pleasing.

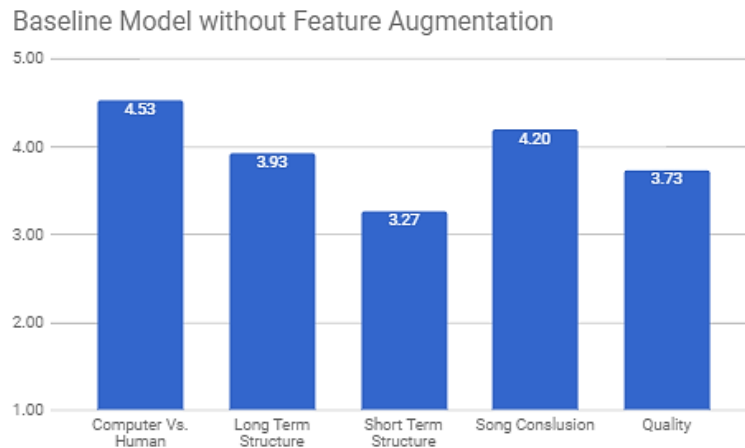


Fig. 6.5 Subjective evaluation of baseline model composition by listeners having musical background (Groups 3 and 4)

6.3.5 Evaluating Counter Augmented Generation

With the introduction of a duration counter for duration features (Modified Counter –R), as seen in Fig. 6.6, the overall feedback appears more positive as compared to the baseline model in Fig 6.5. Listeners scored this sequence's overall quality as **3.20** slightly below average, which is considered a positive sign when compared to base model's score of **3.73**. A slight improvement in long term structure score of **3.53** can be seen as compared to the base model **3.93**. Due to countdown augmentation, it can be seen that listeners scored this sequence much higher than baseline model as the sequence tries to conclude in a more meaningful way, leading to slightly improved over all quality. Song conclusion was thus scored at **2.8** compared with **4.2** for baseline model. The improvement in long term structure could be an effect of better and more natural conclusion which usually is a sign of human compositions.

This leads to conclusion that the countdown augmentation has some positive effect on the overall aesthetic quality of generated music and the subjective evaluation of this model leads to an improved score as compared to the base model and is more pleasant towards listeners.

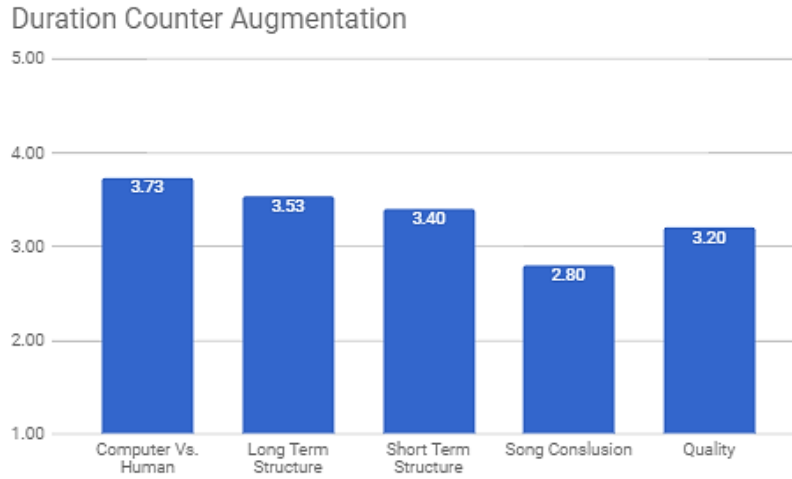


Fig 6.6 Subjective evaluation of countdown augmented model composition by listeners having musical background (Groups 3 and 4)

6.3.6 Evaluating Counter and Metrical Feature Augmented Generation

Finally, the combined effect of duration countdown and metrical information augmentation on the model trained on 4/4 musical sequences (4/4 Data All Feat –R) is examined as shown in Fig. 6.7.

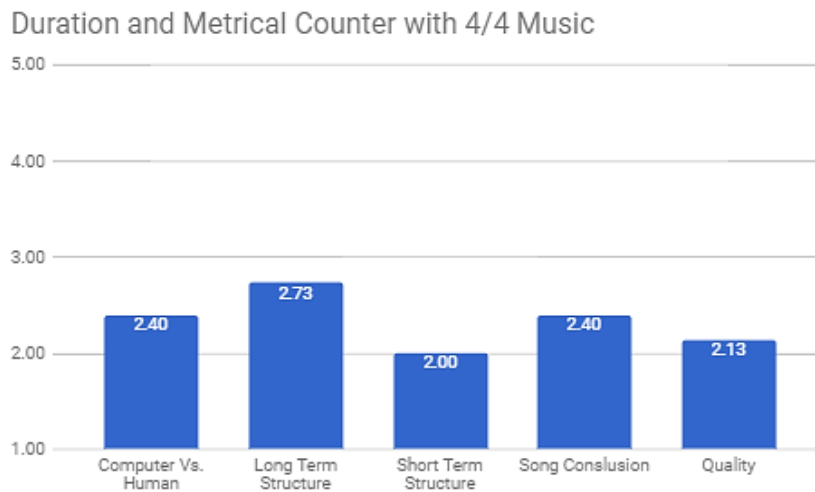


Fig. 6.7 Subjective evaluation of countdown and metrical information augmented model composition by listeners having musical background (Groups 3 and 4)

The final augmentation experiment leads to generation of musical sequences which are structurally more sound and have a much improved aesthetic appeal. The sequence was identified as having more human-like qualities than a computer-generated music showing an above average score of **2.4**. The sequence was scored better due to improved short-term structure with a score of **2.0**. Also, the generative model maintains the key for most parts and rhythm through most of its duration. The model learnt to introduce more realistic short term note transitions which are normally associated with human compositions. The sequence's long-term structure was scored slightly below average with **2.73**. Sequence conclusion was scored

at **2.4** which is better than seen with counter only augmentation. This could be due to the fact that model maintains the rhythm and key along adjacent bars of music and attempts to conclude the song with the dominant note seen in preceding bars and comes to a conclusion in a pleasant manner. This behaviour resulted as an improved overall quality of sequence.

Due to largely reduced responses, it was decided to stop this analysis at this stage and plan for future experiments and engaging more domain experts for productive feedback and constructive criticism of generated melodies.

*A zip file containing detailed user responses for each question can be found in the digital **Appendix E.4_subjective_evaluation_responses.zip** (long html file, best viewed with chrome browser). A spreadsheet with data extracted from these tests is also being attached as **Appendix E.5_Listening_tests_data.xlsx**. Due to length and nature of these documents, they are being presented in a digital format.*

Chapter 7. DISCUSSION

This chapter provides a brief discussion around the research objectives set in Chapter 1 and summarizes the findings. The chapter also highlights the products that are being delivered as a part of this submission, and as listed in Chapter 1. A brief summary of new findings from objective and subjective evaluations from Chapter 5 and Chapter 6 is also provided.

7.1 FULFILMENT OF RESEARCH OBJECTIVES

Following passage lists all the objectives defined at the start of the project and a quick discussion on the status of achievement in each case.

Literature Review and Development Framework

The first objective set for this research internship was to research music generation literature and deep learning frameworks to identify state of the art approaches. A number of different deep learning frameworks were considered and experimented with, identifying TensorFlow with better resources, examples and support as compared to other platforms. The availability of required resources helped greatly towards understanding the relevant concepts in a limited time. This allowed author to develop of a sound theoretical framework of deep learning and gain hands-on experience of developing deep generative neural models, feature engineering in neural networks and generation techniques. It can be confidently said that the first objective was successfully achieved.

Developing Baseline Model

The baseline model, used for this experiment was heavily inspired from dual SoftMax classification approach for melody and harmony (Yoav, 2016). The code base was initially found to be very complicated due to complex pre-processing and encoding routines developed in NumPy and Python-midi. Some experimenting with the code allowed better understanding and with slight modification to Yoav's experiment, the code base proved to be an excellent starting point for further experiments. The baseline model developed from this code offered a high level of flexibility and exploration towards proposed experimentation, leading to fulfilment of second objective.

Creating feature augmented datasets

A number of datasets were created during pre-processing stage with different combinations of augmented features, used within training and validation and are being submitted with this report. Creating these datasets allowed fulfilment of third stated objective and these datasets were successfully used in the experiments highlighted in this report.

Model performance evaluation

Table 5.6 show that augmenting extra structural and metrical information with input encoding positively affects the performance of the models. The song duration counter augmentation experiment showed a clear improvement in the predictive ability of the model with lower loss values as shown by results. The metrical feature augmentation did not help too much towards

loss improvement. However, as stated, the subjective understanding of music may not directly relate to improvement in model loss as seen in subjective evaluation of sequences. No long-term structure augmentation was explicitly tried for these experiments due to lack of labelled data in this domain, Result confirms that the effectiveness of chosen approaches and methods targeting short term structure and duration control. It can hence be stated that this objective was largely met, with a caveat. In order to scientifically present the results as significant and successful, further experimentation with improved infrastructure and larger datasets are required.

Sequence generation with feature augmentation

Augmenting metrical information to music of matching metre helped improve the short-term structure of music as evident from the piano-roll representation of the output of final experiment in Fig. 5.14. The final model (StructLSTM) shows an increased focus on the root note along adjacent bars and attempts to introduces more elaborations and expression like in the melody as compared to base model. The harmonic structure of generated sequences also shows a clear improvement over base model and the model attempts to maintain the metre and rhythm over short-term during generation phase. Being able to achieve generation from structure augmented model was considered a fulfilment of this objective

Subjective Evaluation

The subjective evaluation of sequences that included the duration feature showed a higher score for song conclusion when compared to the baseline model. This behaviour was witnessed for all the sequences generated with counter augmented models. Metrical feature augmentation evaluation also showed the model's ability to help maintain a better short-term structure and enhance the overall aesthetic quality of the sequence. No major improvement in the long-term structure was witnessed. A slightly improved score in the final experiment was witnessed which was thought to be mainly a result of an improved song conclusion followed by a well formed short term structure. This understanding reflects the fulfilment of this objective

7.2 NEW FINDINGS

Based on the experiments conducted along with objective and subjective evaluation of results, a number of general conclusions were drawn. Although these conclusions have been discussed throughout the results section, these are summarized here as the findings of this research project

Duration Counter and Song Conclusion

Augmentation of a duration counter can help conclude a song in a more pleasant and aesthetic way. Generated melodies learn conclusion of each song through this counter and attempt to imitate a similar behaviour during the generation phase, based on the notes played in the concluding bars of music.

Control of song duration

Augmenting a duration counter allows the network to generate and conclude a sequence independent of number of generative cycles defined. This was shown as experiments by

changing counter values with respect to total generation steps. The model does not generate any further note transitions once the conclusion is reached i.e. the counter reaches zero.

Improvement through metrical knowledge augmentation

It was also shown that metre augmentation leads to further improvements in short term structure of generated sequence. This improvement can be seen and heard as the final model attempts to maintain the metre and rhythm in most part of the generated sequences. A number of casual listeners reported these sequences as sounding very natural or having human-like expressive elaborations.

Improved loss value and quality of generated sequence

It was demonstrated that musical concepts including key and rhythm cannot be fully understood and ensured only through achieving improvement in model loss. A better loss value may not directly relate to its aesthetic quality. Music has a complex subjective expression which cannot be directly modelled. Feature augmentation used in this case and other similar techniques can be tried in this regard.

7.3 PRODUCTS DELIVERED

As listed in the 1st Chapter, following products are being delivered with this report. Some of the products can be accessed online, however they are all being submitted with this report, and attached in digital format where applicable.

- A pre-processing library for extracting melody, harmony and additional structural features from a MIDI corpus. This is being submitted as python code with slight modification to midi parsing from (Yoav, 2016), as detailed in methods and approaches.
- A set of methods for feature engineering and augmentation of structural information aimed at providing domain knowledge to the model. This methodology has been highlighted at section 4.5 of this report dealing with the motivations and methods chosen to develop a feature augmentation strategy.
- A feature engineering and augmentation framework that is applicable to both training and generation phases. The “**preprocess.py**” file contains necessary steps and a feature augmenting routine to extract structural information, create engineered features and augment these features to the datasets. The “preprocess.py” file is attached as **Appendix C.1** and generated datasets are also being submitted in digital format with this report as a zip file titled **Appendix D.1_datasets.zip**.
- A structure augmented, generative deep neural machine that would facilitate musical knowledge representation in a deep learning paradigm. A number of trained networks, highlighting all the experiments mentioned in this report are being submitted with the report as a digital submission **Appendix D.2_Trained_Models.zip**. These appendices and can be viewed at the GitHub repository dedicated to this project¹ along with other relevant resources.

¹ StructLSTM (<https://github.com/ShakeelRaja/structlstm/tree/master/Appendices>)

- Performance evaluation of the proposed system against base model using standard network loss calculation. Chapter 5 shows performance evaluation and comparisons between different models augmented with proposed engineered features. This includes base model and a number feature augmented approaches as listed earlier. A set of musical sequences generated as a result of different augmentation experiments. The musical sequences used within these experiments are available online on YouTube links provided in section 6.2, and are also being submitted in mp3 format with this report. These sequences can be accessed via the online listening platform listed in this section, however, the sequences appear in a random order without any labels. The audio samples are being attached to this report as additional submission in ***audio_samples.zip***.
- A subjective evaluation conducted through Turing style listening test to measure aesthetic improvement for generated music. The subjective evaluation test can be accessed via the link provided in section 6.2 and is still on-going. The results of this evaluation have been summarized and discussed in Chapter 6. Additional digital submission ***Appendix_E.4_Subjective_eval_responses.zip*** contains detailed user responses, and ***Appendix_E.5_Listening_test.data.xlsx*** is spreadsheet of data extracted from these tests.

This leads to the conclusion that all the products that were intended at the initial stages of the project have been successfully developed and delivered.

7.4 RESEARCH QUESTION

The original research question set in chapter was as follows

"Can we improve the performance of a Deep Recurrent Neural Network by augmenting engineered structural features to the training process in a Language Modelling framework, in order to stochastically generate musical sequences with improved structure and to achieve better control over the generative process"

Based on the discussion provided in this chapter, it is claimed that the research question was successfully answered in positive, as the results clearly showed that an RNN model can be augmented with extra structural knowledge during training and generation to improve its performance towards model loss as well as towards producing more pleasant music.

CHAPTER 8. EVALUATION, REFLECTION AND CONCLUSION

This chapter covers the evaluation of project planning, research objectives and approaches used to meet these objectives. Followed by author's reflection on learning and a conclusion.

8.1 EVALUATION OF METHODS AND APPROACHES

Changes in original project proposal

As mentioned in the Introduction, there was a change of research objectives from development of a Neural Symbolic System to structural feature augmented Recurrent Neural Networks. The initial set of objectives aimed at augmenting grammar based musical rules into the learning process for structured music generation. The literature review conducted towards writing the proposal showed a great potential in Neural Symbolic approach, however, the complexity and challenge towards augmenting musical grammars into exiting deep learning platforms could not be clearly comprehended. The temporal associations between musical events appear to have both short term and long-term relationships with each other. Logic Tensor Network (Serafini & Garcez, 2016), or LTN, which was initially suggested for implementing this approach currently lacks the ability to work with temporal data. Applications of rules for time-series data into an RNN requires some sort of memory management technique allowing the network to remember event associations over long term and short-term structure. An implementation of this idea has been shown with Neural Turing Machines developed by Alex Graves in (Graves, Wayne & Danihelka, 2014). However, due the complexity of this system, it was decided to use other means of knowledge augmentation for this project. The idea of structural knowledge augmentation into network input encoding was given by the project supervisor, Dr Tillman Weyde who is an active researcher in this domain. The new objectives set under this approach were largely met as shown in the Results and Discussion section.

Literature Review

For the author, this project was a first experiment in generative music and deep learning, hence involved an extensive literature search and review and thorough investigation into deep learning frameworks. Music generation with deep learning is a growing and rich area of interest, and most of the noteworthy experiments in this domain appeared in last two to three years, resulting as a lack of domain specific literature. Most of the literature from this area at present appears either conference papers or exist as on-line articles and tutorials. The simple LSTM based approach proposed by Eck (Eck & Schmidhuber, 2002) is still being heavily employed by most experiments in terms of encoding, data representation generation techniques with RNNs and symbolic music data. Some current experiments have attempted to improve model performance through improved loss functions (Yoav, 2016), adding statistical constraints towards generation (Sun et al., 2016) and learning music dynamics for a higher expression in generated music (Malik & Ek, 2017).

This research project attempts to contribute towards these on-going developments by suggesting a novel way of structural feature augmentation for an improved music learning and generation performance of an LSTM model.

Development Platform

A number of deep learning platforms were considered for this project. Some of these frameworks, mainly PyTorch offered great flexibility and experimentation potential, and might have proven to be more beneficial than TensorFlow towards deep learning for music generation. However, a lack of resources, literature and time limitations for this internship did not permit experimentation with it. Currently PyTorch is heavily being employed for NLP researchers and has not been tried in generative music domain. TensorFlow, on the other hand, has a rich set resources offered by Google and other expert websites so the decision was made to work with TensorFlow for the duration of this project. The Author has plans to port the developed code base to PyTorch to replicate this experiment as a Big Data project through a many-fold increase in training data, utilizing GPU computation and Dynamic Computational Graphs offered by PyTorch.

Experiments

During this project, three main experiments were planned to evaluate the effectiveness of chosen approaches. Developing the base model, augmenting a song duration counter, and augmenting metrical information as extra structural knowledge to the base model. Developing the base model was mainly considered a learning objective for the author to be able to conduct further experimentation. it was decided to search exiting experiments that might be able to provide a functional code base. A number of such published and unpublished experiments were investigated and the decision was made to try the proposed approach with Yoav Zimmerman’s Dual soft-max classification model (Yoav, 2016). Although this work is not published, it provided an excellent starting point for further experiments. A justification for choosing this experiment is provided in section 4.2. The code base, developed in Python 2.7 with TensorFlow 0.8 which is only available on Linux platform which posed some limitations later during experiments. It was found very challenging to run this code for cross validation due to CUDA compatibility issues faced towards running this code on University’s GPUs based deep learning setup, which needs later versions of both Python and TensorFlow. Mainstream MIDI pre-processing libraries like Python-midi and Mingus used within this project are not fully compatible with newer versions of Python. This did not affect with the experimentation and no major issues were faced while working in this environment. Scalability of this experiment, however, remained limited towards increasing data size or running more rigorous tests.

Training Data

The Nottingham dataset, used for training and validation within this experiment is compact in size. Due to lack of a labelled dataset for the experiment, the structural information available in MIDI headers was extracted and engineered. MIDI datasets lack information on musical phrases found within a composition as it only stores time-based events. The need for a better labelled data was hence felt which could have allowed learning musical phrases found across bars in a composition. At present, the only such dataset available is GTTM, developed by Hamanaka (Hamanaka, Hirata & Tojo, 2015). This dataset, although labelled in great details, was not considered suitable for deep learning as it only contains about 300 melodies.

Feature Augmentation Approach

Effectiveness of augmenting structural features was confirmed by an increase in predictive ability of the network. The final model shows an improved predictive performance. The model shows an improved short-term structure, the melody attempts to maintain the rhythm and key along neighbouring bars due to metrical augmentation and model attempts to conclude a song in a much more natural fashion as compared to the base model. This approach shows great potential and invites further experimentation to add carefully engineered features and datasets covering different musical styles.

Model performance

The objective evaluation comprised comparisons of loss values, training time and visual detection of musical structure from piano roll representations leading to an objective conclusion. However, cross validation could not be performed to measure the statistical significance of the results when comparing the performance of feature augmented models to baseline model. This was mainly due to time and computational constraints. Significance testing is planned for future experiments. The author committed to this project with an initial intention to get this work published. In order to identify this approach as scientifically significant, further testing of the models is planned with more data containing more musical styles and better computing resources.

Subjective evaluation

The Turing Style listening tests conducted for this experiment showed a general improvement in the quality of generated music as scored by listeners. However, it was noticed that most of the participants were casual music listeners and some found it hard to relate their subjective experiences to musical terms, mainly short term and long-term structure. Listeners who were not familiar with the area of generated music did not have a clear understanding of the qualitative and structural differences between a computer and a human composition. The results of the survey were therefore kept focused at those listeners who were music professionals, enthusiasts, played some instrument as a hobby or knew basics of music theory. This hugely reduced the number of participants from 40 to 15. It was felt that a more in-depth listening test consisting of individuals who have knowledge of music and deep learning would have been more beneficial. However, it was considered a difficult task to achieve within the limited timeframe available for this survey at the last stages of this project. No further testing of the results was performed for identifying significance of the results due to very limited data. At present, the results are compiled as graphs allowing visual analysis of frequency distributions. This shows room for improvement towards developing test design that allows casual listeners to better score the sequences, and engaging more domain experts for productive feedback

8.2 FUTURE WORK

As described earlier, it is planned to scale up this experiment by increasing the size of training data and porting the code to an efficient framework allowing GPU utilization. Some parts of the code need improvement as the current pre-processing is set around MIDI files having two tracks with melody and harmony notes. A lot of free MIDI resources are currently available

online but they may not necessary follow this assumption. A more flexible pre-processing routine will be formulated to detect the type of MIDI data and processing it accordingly. The code will also need some modifications towards model development suitable for PyTorch framework that uses dynamic graphs as compared to TensorFlow's static graphs. A more rigorous testing apparatus employing cross fold validation for significance testing will be implemented. Listening tests currently performed lack expert feedback and thus require experts to carefully analyse and score generated music.

The chosen approach will also be tried with other more sophisticated RNN architectures including bi-directional RNNs, bi-axial RNNs, clock-work RNNs and tree RNNs. Tree RNNs, offered by PyTorch allow definition and augmentation of a tree hierarchy with the input data which can be useful towards describing the global structure of a sequence and how individual music events in the sequence are related to it. The capability of this experiment to produce a meaningful short-term structure with Tree RNN's ability to develop a global structure can lead to generation of a more human-like composition with a deeper subjective meaning, which is the primary objective of generative music domain.

The author has extensive experience of Indian classical music which is very strongly bounded by compositional rules and rhythmic scales. A lot of literature is currently available for devising these rules and building a composition around them. Some literature uses a piano-roll like format to explain these rules and to group melodies around specific rhythmic scales. However, most of this literature is currently in published format and written in Urdu/Hindi languages and hence not directly understandable by computational algorithms as well as MIDI or a piano roll format. The author has future plans to use deep learning with NLP based language translation techniques to convert the limited vocabulary used within this literature to a standard piano-roll format. This can potentially allow the model to understand note transitions based on given rules which can be treated as structural features and can be augmented using the approaches shown in this report. This can effectively solve the problem of lack of labelled data and the lack of global structure in generated melodies.

8.3 REFLECTION

This project has been a very exciting and insightful venture into the domain of deep learning and generative arts. The research allowed me to come to an intersection of areas that I have always felt passionate about, including music, technology and creative thinking. The extensive learning experience has helped me to further broaden my horizons in the field of Computer Science and Artificial Intelligence. Due to some difficulties at the early stage of this project, mainly the demise of author's mother, and later change in research objectives, a simplified approach was chosen to show this project as a proof of concept. Furthermore, there was some room for improvement identified during both objective and subjective evaluation stages. These experiences, however provided a lot of learning and understanding around research issues and risk planning.

I intend to further develop this approach and present it as a PhD proposal as I strongly felt that due to complex nature of both music and deep learning, more time is required to develop

models that attempt to achieve state of the art performance. A detailed framework needs to be developed and new datasets need to be generated in order for enhancing model's generalizability.

I strongly believe that this internship has successfully equipped me with a theoretical foundation, tools and technologies that are considered state of the art towards AI and machine learning, and has enhanced future career prospects for me.

8.4 CONCLUSION

Findings of this research show a lot of potential for chosen methods and approach to be employed within generative music applications used by technology driven businesses involved in using artificial intelligence to create custom music tracks for their clients. The ability to maintain song rhythm and metre can be used effectively in real time applications for musical accompaniment and to aid professional musicians who employ state of the art music technology for their performances.

A number of such experiments have recently appeared on Google AI experiments ¹, mainly an experiment called AI Duet, that learns the scale of the keys played by a musician and based on training data, generates melodies while attempting to maintain the scale and rhythm defined by the musician in real time, thus creating a duet like performance with the musician. A number of software and hardware based digital instruments used within modern production studios have attempted with introducing similar techniques to aid compositional work. Korg Karma engine² generates musical phrases based on the key and rhythm set by the user. Researchers at Sony's AI lab in have released world's first ever AI based pop music in raw audio through Flow Machines platform³.

This research shows the potential towards the ability to generate music with different styles once trained on the large enough data that cover these styles and structural definitions of these styles that can allow network to learn different variations found in different styles and genres. The simplified approach, tried with a simple dataset for this project indeed shows a potential to be developed into a commercial system, such as ones described above.

As an overall conclusion, it is stated that this research has been an exciting and insightful learning experience for the author. The research objectives were successfully met and the research question was positively answered. The research has provided an opportunity to get hands on practice with modern deep learning tools and has opened new research directions for future.

¹ Google AI experiments (<https://experiments.withgoogle.com/ai>)

² Korg Karma (<http://www.karma-lab.com/korgkarma/korgkarma.html?p=korgkarma/index.html>)

³ The first pop song by AI (<https://qz.com/790523/daddys-car-the-first-song-ever-written-by-artificial-intelligence-is-actually-pretty-good>)

The code developed and modified for this research project is available at the GitHub repository at <https://github.com/ShakeelRaja/structlstm>.

Digital appendices can be accessed at <https://github.com/ShakeelRaja/structlstm/tree/master/Appendices>

The on-going listening test can be accessed on-line at <https://goo.gl/J24GVq>

Audio samples (attached as **audio_samples.zip** file) used for listening tests can also be accessed directly via YouTube links provided, or downloaded from https://github.com/ShakeelRaja/structlstm/tree/master/audio_samples

GLOSSARY OF ACRONYMS

AI	Artificial Intelligence
ANN	Artificial Neural Network
BPTT	Back-Propagation Through Time
GAN	Generative Adversarial Network
GD	Gradient Descent
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
MIDI	Musical Instruments Digital Interface
ML	Machine Learning
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
NN	Neural Network
RBM	Restricted Boltzmann Machine
ReLU	Rectified Linear Unit
RL	Re-enforcement Learning
RNN	Recurrent Neural Network
StructLSTM	Structure Augmented Long Short-Term Memory (Network)

REFERENCES

- Abdallah, S.A. and Gold, N.E., 2014. Comparing models of symbolic music using probabilistic grammars and probabilistic programming.
- Aleksander, I. and Morton, H., 1990. *An introduction to neural computing* (Vol. 3). London: Chapman & Hall.
- Bengio, Y., Courville, A.C. and Vincent, P., 2012. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR, abs/1206.5538*, 1, p.2012.
- Baroni, M. and Callegari, L. eds., 1984. *Musical grammars and computer analysis*. Florence: Olschki.
- Baroni, M., Maguire, S. and Drabkin, W., 1983. The concept of musical grammar. *Music Analysis*, 2(2), pp.175-208.
- Biles, J.A., 1994, September. GenJam: A genetic algorithm for generating jazz solos. In *ICMC* (Vol. 94, pp. 131-137).
- Boulanger-Lewandowski, N., Bengio, Y. and Vincent, P., 2012. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*.
- Briot, J.P. and Pachet, F., 2017. Music Generation by Deep Learning-Challenges and Directions. *arXiv preprint arXiv:1712.04371*.
- Briot, J.P., Hadjeres, G. and Pachet, F., 2017. Deep Learning Techniques for Music Generation-A Survey. *arXiv preprint arXiv:1709.01620* (pp. 57-58).
- Chung, J., Gulcehre, C., Cho, K. and Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.
- Coca, A.E., Corrêa, D.C. and Zhao, L., 2013, August. Computer-aided music composition with LSTM neural network and chaotic inspiration. In *Neural Networks (IJCNN), The 2013 International Joint Conference on* (pp. 1-7). IEEE.
- Cope, D. and Mayer, M.J., 1996. *Experiments in musical intelligence* (Vol. 12). Madison, WI: AR editions.
- Davismoon, S. and Eccles, J., 2010, April. Combining musical constraints with Markov transition probabilities to improve the generation of creative musical structures. In *European Conference on the Applications of Evolutionary Computation*(pp. 361-370). Springer, Berlin, Heidelberg.
- Deng, L. and Yu, D., 2014. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4), pp.197-387.
- Denton, E.L., Chintala, S. and Fergus, R., 2015. Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks. In *Advances in neural information processing systems* (pp. 1486-1494).
- Diaz-Jerez, G., 2011. Composing with Melomics: Delving into the computational world for musical inspiration. *Leonardo Music Journal*, pp.13-14.
- Dietterich, T.G. and Michalski, R.S., 1986. Learning to predict sequences. *Machine learning: An artificial intelligence approach*, 2.
- Ebcioğlu, K., 1988. An expert system for harmonizing four-part chorales. *Computer Music Journal*, 12(3), pp.43-51.
- Eck, D. and Schmidhuber, J., 2002. A first look at music composition using lstm recurrent neural networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, 103.
- Eck, D. and Schmidhuber, J., 2002. Finding temporal structure in music: Blues improvisation with LSTM recurrent networks. In *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on* (pp. 747-756). IEEE.
- Fernández, J.D. and Vico, F., 2013. AI methods in algorithmic composition: A comprehensive survey. *Journal of Artificial Intelligence Research*, 48, pp.513-582.

- Fiesler, E., 1996. Neural network topologies. *The Handbook of Neural Computation*, E. Fiesler and R. Beale (Editors-in-Chief), Oxford University Press and IOP Publishing.
- Graves, A., Wayne, G. and Danihelka, I., 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.
- Gregor, K., Danihelka, I., Graves, A., Rezende, D.J. and Wierstra, D., 2015. DRAW: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*.
- Groves, R., 2016. Automatic Melodic Reduction Using a Supervised Probabilistic Context-Free Grammar. In *ISMIR* (pp. 775-781).
- Hamanaka, M., Hirata, K. and Tojo, S., 2015, June. σ GTTM III: Learning-Based Time-Span Tree Generator Based on PCFG. In *International Symposium on Computer Music Multidisciplinary Research* (pp. 387-404). Springer, Cham.
- Hadjeres, G. and Pachet, F., 2016. DeepBach: a Steerable Model for Bach chorales generation. *arXiv preprint arXiv:1612.01010*.
- Haykin, S., 2004. A comprehensive foundation. *Neural Networks*, 2(2004), p.41.
- High, R., 2012. The era of cognitive systems: An inside look at ibm 77acili and how it works. *IBM Corporation, Redbooks*.
- Hiller, L.A. and Isaacson, L.M., 1959. Experimental music: composition with an electronic computer
- Hiraga, R., Bresin, R., Hirata, K. and Katayose, H., 2004, June. Rencon 2004: Turing test for musical expression. In *Proceedings of the 2004 conference on New interfaces for musical expression* (pp. 120-123). National University of Singapore.
- Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. *Neural computation*, 9(8), pp.1735-1780.
- Hornik, K., Stinchcombe, M. and White, H., 1989. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5), pp.359-366.
- Horner, A. and Goldberg, D.E., 1991. Genetic algorithms and computer-assisted music composition. *Urbana*, 51(61801), pp.437-441.
- Ioffe, S. and Szegedy, C., 2015, June. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning* (pp. 448-456).
- Jaques, N., Gu, S., Turner, R.E. and Eck, D., 2017. Tuning recurrent neural networks with reinforcement learning. *arXiv preprint arXiv:1606.01541*
- Johnson, D., 2015. *Composing Music with Recurrent Neural Networks*. [ONLINE] Available at: <http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/>. [Accessed 1 January 2018].
- Karpathy, A., 2015. *The Unreasonable Effectiveness of Recurrent Neural Networks*. [online] Karpathy.github.io. Available at: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/> [Accessed 14 Jan. 2018].
- Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).
- Langacker, R.W., 1987. *Foundations of cognitive grammar: Theoretical prerequisites* (Vol. 1). Stanford university press.
- Lavrenko, V., 2018. *Backpropagation, How it works*. [Online Video]. 31 August 2015. Available from: <https://www.youtube.com/watch?v=An5z8lR8asY>. [Accessed: 8 January 2018].
- LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. *Nature*, 521(7553), pp.436-444.
- Lerdahl, F. and Jackendoff, R., 1985. *A generative theory of tonal music*. MIT press.
- London, J., 2012. Hearing in time: Psychological aspects of musical meter. *Oxford University Press*.

- Malik, I. and Ek, C.H., 2017. Neural translation of musical style. *arXiv preprint arXiv:1708.03535*.
- Mozer, M.C., 1994. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, 6(2-3), pp.247-280.
- McCulloch, W.S. and Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4), pp.115-133.
- Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H. and Ng, A.Y., 2011. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)* (pp. 689-696).
- Nierhaus, G., 2009. *Algorithmic composition: paradigms of automated music generation*. Springer Science & Business Media.
- Papadopoulos, G. and Wiggins, G., 1999, April. AI methods for algorithmic composition: A survey, a critical view and future prospects. In *AISB Symposium on Musical Creativity* (Vol. 124, pp. 110-117). Edinburgh, UK.
- Pascanu, R., Mikolov, T. and Bengio, Y., 2012. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063.
- Pearce, M. and Wiggins, G., 2001. Towards a framework for the evaluation of machine compositions. In *Proceedings of the AISB'01 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences* (pp. 22-32).
- Quick, D., 2014. *Kulitta: A framework for automated music composition*. Yale University.
- Serafini, L. and Garcez, A.D.A., 2016. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. *arXiv preprint arXiv:1606.04422*.
- Sarroff, A.M. and Casey, M.A., 2014. Musical audio synthesis using autoencoding neural nets. In *ICMC*.
- Schmidhuber, J., 2015. Deep learning in neural networks: An overview. *Neural networks*, 61, pp.85-117.
- Scholes, P., 1977. Metre and Rhythm.
- Siegelmann, H.T. and Sontag, E.D., 1995. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1), pp.132-150.
- Sigtia, S., Benetos, E. and Dixon, S., 2016. An end-to-end neural network for polyphonic piano music transcription. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 24(5), pp.927-939.
- Sturm, B., Santos, J.F. and Korshunova, I., 2015. Folk music style modelling by recurrent neural networks with long short term memory units. In *16th International Society for Music Information Retrieval Conference*.
- Sun, Z., Liu, J., Zhang, Z., Chen, J., Huo, Z., Lee, C.H. and Zhang, X., 2016. Composing Music with Grammar Argumented Neural Networks and Note-Level Encoding. *arXiv preprint arXiv:1611.05416*.
- Tallal, P. and Gaab, N., 2006. Dynamic auditory processing, musical experience and language development. *TRENDS in Neurosciences*, 29(7), pp.382-390.
- Thom, B., 2000, June. BoB: an interactive improvisational music companion. In *Proceedings of the fourth international conference on Autonomous agents* (pp. 309-316). ACM.
- Tieleman, T. and Hinton, G., 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2), pp.26-31.
- Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. and Kavukcuoglu, K., 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.
- Waite, E. 2016. *Generating Long-Term Structure in Songs and Stories*. [ONLINE] Available at: <https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn>. [Accessed 15 January 2018].
- Walder, C., 2016, November. Modelling Symbolic Music: Beyond the Piano Roll. In *Asian Conference on Machine Learning* (pp. 174-189).

Wang, X. and Wang, Y., 2014, November. Improving content-based and hybrid music recommendation using deep learning. In *Proceedings of the 22nd ACM international conference on Multimedia* (pp. 627-636). ACM.

Xenakis, I., 1992. *Formalized music: thought and mathematics in composition* (No. 6). Pendragon Press.

Yang, L.C., Chou, S.Y. and Yang, Y.H., 2017, October. 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*.

Yoav, Z.. 2016. *Music Language Modeling with Recurrent Neural Networks*. [ONLINE] Available at: http://yoavz.com/music_rnn/. [Accessed 1 January 2018].

Zimmermann, D., 1998, June. Modeling Musical Structures, Aims, Limitations and the Artist's Involvement. In *Proc. Constraints techniques for artistic applications, Workshop at ECAI* (Vol. 98).

APPENDIX A

INITIAL INTERNSHIP PROPOSAL

Internship Proposal – MSc Data Science (2016/17)

Neural-Symbolic Music Generation Using Probabilistic Musical Grammars

Shakeel ur Rehman Raja

ABSTRACT

This research proposal looks at design and implementation of a Neural-Symbolic generative model for generating structured musical sequences. The proposed model would facilitate the augmentation of musical knowledge within a Deep Neural Network while learning production probabilities and enforcing structural constraints. The proposal highlights the critical context around the area of research with reference to literature, a research evaluation criterion followed by proposed development framework and a project plan.

25 INTRODUCTION

25.2 Purpose

The purpose of this research internship is to design and implement neural computational models for supervised/unsupervised music sequence generation, while imposing a music-theoretic structural control over the generation process using a formal logic system. The proposed research aims to explore and learn whether Music Language Modelling (MLM) based on Natural Language Processing theories can be used for music generation in Deep Neural Networks. The proposed Neural-Symbolic machine would facilitate symbolic knowledge representation and connectionist learning in an attempt to emulate human like cognitive performance. Natural Language Processing tools and frameworks will be used to represent musical knowledge within the proposed system as probabilistic music grammar rules. Learning and Generation processes will be performed using symbolic music data, with a primary focus on musical structure representation. The system would offer parametric control over the generation process to facilitate different styles of music. The performance of the proposed system will be evaluated using statistical measures based on music theory as well as through Turing test style human evaluation and listening sessions. Other standard evaluation methods like cross validation will be for supervised generation models.

25.2 Products

Following products will be presented at the end of this research project:

- A set of Musical Grammar rules encoded into a formal logic system that would define the style of a music genre. Production probabilities would be calculated using different symbolic music datasets.
- A deep network architecture which would be wired and pre-configured to impose structural constraints on generated musical data.
- A grammar augmented, generative Deep Neural-Symbolic machine that would facilitate symbolic musical knowledge representation with connectionist learning.
- Performance comparison of the proposed system with other similar experiments. These comparisons would compare the Neural-Symbolic approach with pure connectionist and symbolic AI systems.

25.2 Beneficiaries

The beneficiaries of this work comprise the following groups:

- Generative music application and systems developers and practitioners interested in finding novel ways to generate human-like music using computational approaches.
- Music informatics researchers and specialists keen to explore theories and practices of musical improvisations and performances through music information analyses and modelling.
- Musicians employing latest technology to aid their compositional work and live improvisatory performances.

1.4 Research Question

The primary research question, that this project attempts to answer is:

“Can we augment a Deep Neural Network architecture with generative musical grammars using a Neural-Symbolic framework, for symbolic musical knowledge representation and connectionist learning, in order to stochastically generate musical sequences while enforcing structural constraints on the generative process”

1.5 Research Objectives

In order to answer above research question, the main objectives proposed for this research project are as follows:

1. How can we best encode and normalize symbolic musical data (MIDI, MusicXML and ABC formats etc.) to be processed and analysed with deep networks in a neural-symbolic domain.
2. How can generative musical grammars be used for defining style specific music theoretic concepts and structural constraints for symbolic knowledge representation within Deep Neural Networks?

3. Can we use a formal logic system to define grammars as logical rules to control melodic sequence generation and constraint definition in deep neural networks?
4. Is it possible to learn production probabilities from symbolic music data using NLP and grammar rules for melodic and harmonic sequence generation and evaluate using cross entropy?
5. Can rule augmented Neural-Symbolic music generation systems improve upon purely connectionist and symbolic AI methods currently being used for music generation?

25 CRITICAL CONTEXT

Algorithmic music composition has been classically described as setting a sequence of rules for solving a problem of combining individual musical components into a complete composition. Alan Turing, widely considered as father of theoretical computer science, conducted the first known computer music experiment in 1951. Illiac Suite, later termed as String Quartet No. 4, was another early experiment using computers for music generation [16]. The term “Generative Music” using computational algorithms was first coined by Brian Eno’s work, identifying it as a broad and rich category within the spectrum of generative arts. The idea revolves around using a computational difference engine to create compositions where every piece of musical is essentially unique and thus eliminating the “clockwork repetitions” found in recorded music [9].

Since the inception of the idea of generative music, there have been a number of attempts to generate music using computational algorithms including Markov Models, Genetic Algorithms, Natural Language Processing, Cellular Automata, Machine Learning and Neural Computing as reviewed by [26] [25] and [10]. These methods have been employed by researchers and musicians with considerable success; however, there still exist many caveats such as lack of domain subjectivity, increased level of system complexity, absence of human-like improvisations, questionable quality of generated music and dependence of large training datasets and complex composition rules.

2.1 ALGORITHMIC MUSIC GENERATION

Following section provides a brief overview of intelligent music generation techniques that have been employed by researchers and practitioners in the past with selected examples.

Mathematical models and stochastic processes, mainly Markov chains have been used extensively for music generation since 1950s [16]. Their ability to perform transitions in discrete time steps with a lower level of complexity allows them to be effectively utilized within real time music generation. Interactive Jazz improvisations with BoB system [29] and simulated annealing to generate rhythm and melody [6] can be seen as natural evolution such models. These models, however, lack the ability to deal with higher abstractions found in human music and need large amounts of data for probabilistic computations to become generalizable.

Knowledge Based Systems (KBS) are primarily symbolic in nature having explicit rules and structural constraints to control the output of the model. Their main advantage is their ability of explain choices and actions taken to generate a certain output. The rule based harmonization system, CHORAL [8] and Intention- based music experiment [32] can be seen as examples of KBS for generating compositions. The main limitations for such systems are an increased level of complication towards knowledge elicitation and a high dependency on the skills of experts. **Evolutionary / Genetic Algorithms** have proven to be an effective alternative to some of techniques mentioned above due to their ability to work with large search spaces and provide multiple unique solutions which a desirable function while dealing with music generation. Genetic programming has been proved to generate programs that produce rhythmic melodies as output when given a melody as an input. One of the early examples in this regard is [17] where authors implemented evolutionary methods for musical thematic bridging. Also, professional score writing with Iamus [7] can be seen as a good candidate example. These methods were capable of producing aesthetically pleasing melodies but lack subjective elements and exhibited lower efficiency due to “Fitness Bottleneck” problem.

Musical Grammars are sets of structural rules to expand high level symbols into detailed sequence of melodies. Recent studies show a strong relation between music and spoken language in terms of their structural representation (Brown et al, 2006). A probabilistic context-free grammar (PCFG) is a context free grammar (CFG) modified to give each rule a probability of being used in the output. Baroni (Baroni et al., 1984) developed a grammar for generating music in the style of Lutheran chorales. The logical probabilistic framework, PRISM [1] for automatic music analysis and reduction, GTTM Analyser [15] for unsupervised PCFG based music generation, and automatic melodic reduction and generation [14] with supervised learning of PCFGs can be presented as key examples in this context. Another type of musical grammar called Probabilistic Temporal Graph Grammars (PTGGs), have been implemented in a generative system Kulitta [27]. PTGGs incorporate both PCFGs and as well as support for high level music structures. This grammar incorporates both metrical structure and pattern repetition by using “Chord Spaces” and genre-specific improvisations towards harmonic and melodic music sequences. Some limitations of grammars have been highlighted as being computationally expensive, over-all questionable quality of an infinite number of generated melodies and expertise needed towards defining music theoretic concepts as grammar rules.

Artificial Neural Network Models (ANNs) have gained special attention by musicians and music informatics researchers as these can be trained on complex structured musical patterns present in an a given musical dataset and generate structured melodic content with improved realism, better control over model definition and generation processes, improvisation and higher flexibility in design [10]. Early experiments with ANNs attempted to compose monophonic melodies based on probabilistic modelling. In recent years, Deep Neural Networks, especially Recurrent Neural Networks (RNNs), as shown in Fig 1, have gained particular popularity in the field of music generation as they allow representation of complex

musical structures including polyphonic generation with temporal structural dependency controls.

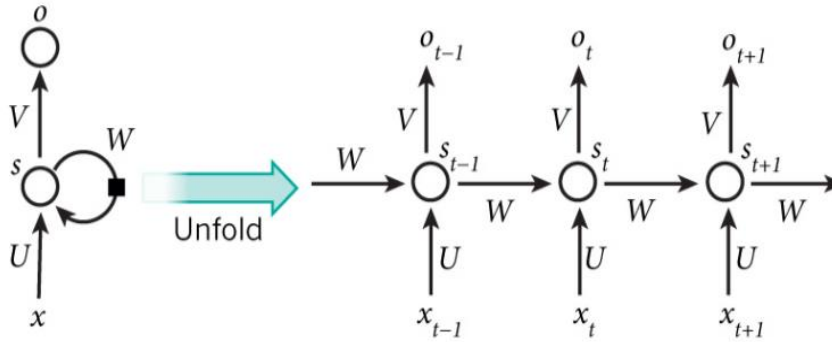


Figure 1: Recurrent Neural Networks make use of sequential information. RNNs perform same task for every element of the given sequence which makes them ideal to calculate the structural and temporal dependencies in sequences of musical data. (Source: Nature)

Some noteworthy mentions in this regard are Long Short-Term Memory Networks (LSTMs) for long term structural dependency control [24], Polyphonic music generation with temporal dependencies using Restricted Boltzmann Machines [3] and Convolutional Generative Adversarial Networks for symbolic learning and generation [30]. Google Magenta, a project of Google brain team (<http://magenta.tensorflow.org>), is the current state of the art for developing machine learning models for generative arts including images, video and music. One noteworthy project under this platform is re-enforcement deep learning for music generation with RL-TUNER [20] to improve the structure and quality of sequences generated by an RNN.

Hybrid Systems for music generation can be explained as systems which are a combination of two or more computational approaches mentioned above [26]. A number of such experiments have been conducting and experimented for music generation with considerable success e.g. Grammars with genetic programming, ANNs with evolutionary algorithms and Rule-based Markov chains as explained by [10]. Neural networks can be configure as hybrid Neural-Symbolic systems by incorporating traditional connectionist learning with symbolic reasoning using propositional or first order logic rules. This approach allows development of efficient models while offering augmentation, reasoning and extraction of domain knowledge at a symbolic and conceptual level. [11]. CILP (Connectionist Inductive Learning and Logic Programing) system proposed by Garcez shown in Fig. 2 is an example of such a Neural-Symbolic framework.

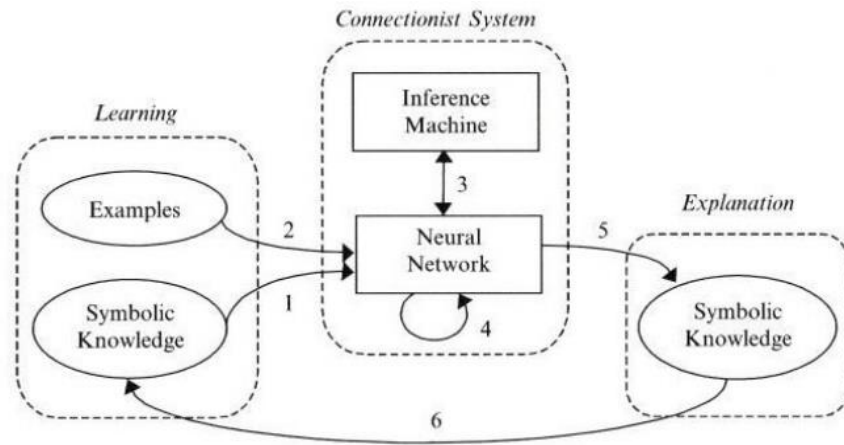


Figure 2: CILP (Connectionist Inductive Learning and Logic Programing) Neural-Symbolic system offers an ANN based model that integrates inductive learning from background knowledge, with deductive reasoning. Source: ([11])

Neural-Symbolic music generation has been experimented in the past by Mozer with CONCERT system (Mozer, 1994). Grammar based symbolic music generation, primarily emerging from GTTM (Generative Theory of Tonal Music) developed in 1980s (Lerdahl and Jackendoff, 1985) allows NLP based Context Free Grammars (CFGs) for probabilistic modelling of musical structure for identifying relationships between melodic components, repetition, transformation and note-variation, and to enforce music-theoretic ideas such as scale, harmony and tonality etc. [1] [14].

2.2 Deep Learning with Logical Reasoning

Integrating domain knowledge as logical rules into gradient based connectionist systems can enable effective knowledge representation, reasoning and extraction in deep neural networks for data driven machine learning applications. The domain knowledge represented in such systems can help identify constraints towards the generative process and can be modified with incoming data to improve constraint reasoning and inference of new rules. Such systems use propositional logic, first/second and higher order logic formalisms for integrating existing knowledge bases into neural networks by changing the weights of neural connections. A number of such frameworks that allow integration of logical formalism into Deep neural have been presented showing considerable improvement towards problem solving in the areas of computer vision, natural language processing, Time series Analysis and other sequential temporal data.

Logic Tensor Networks (LTN) [12] is a framework built for Google Tensor-Flow and uses a variant of first order logic called Real Logic that can reflect truth values as probabilities and semantics defined on real numbers. This allows rule based deductive reasoning and effective relational machine learning. TensorLog [5] is a probabilistic differential database framework which uses logical reasoning with a differentiable process by employing functions that use factor graphs and belief propagation to perform inference in complex logical systems by interrelated clauses and predicates. Another similar approach has been presented by [18] to implement first order logic rules in convolutional and recurrent deep neural network models.

These frameworks have achieved substantial improvements and have managed to achieve state of the art performance as compared to purely connectionist networks and traditional symbolic AI systems. A statistical Music Language Model (MLM) built upon these frameworks can be effectively used for modelling symbolic music sequences that can estimate the distribution of harmonic and melodic transitions from the given musical corpus as accurately as possible. A language model augmented recurrent neural network can estimate the probability distributions over sequence of melodic and harmonic symbols and estimates the likelihood different notes using NLP methods while generating the output.

25 **METHOD AND APPROACHES**

The methodology considered for this project revolves around generation of melodic and harmonic musical sequences in a grammar augmented Neural-Symbolic domain while using different labelled and unlabelled datasets, open-source development tools and standard evaluation criteria as highlighted in the following text.

3.1 Literature Review

The literature search and review for this proposal is performed using Google Scholar and the City University Library to identify the proposed research question and to plan the methodology. An ongoing review during planning, experimentation and evaluation phases will be made of other text related to the topic. It must also be mentioned that some of papers studied and presented in this regard are recent conference papers which have not been published in a journal due to their recency.

3.2 Software and Tools

- Python Programming Environment Continuum Analytics' Anaconda based distribution of Python programming language will be used in this experiment as primary programming environment. This environment is selected due to large community support and seamless integration with all major Python libraries.
<https://www.continuum.io/>
- Natural Language toolkit (NLTK) Library. NLTK library will be used for building symbolic probabilistic grammar rules to apply on data in ABC symbolic notation.
- Tensor Flow/Magenta Deep Learning Libraries. Tensor Flow and Magenta open source libraries by Google will be used to build and train neural networks for melodic generation. This learning environment is selected due to its offerings on a variety of deep neural network architectures and active user community. A vast number of current research activities into the field of music generation are being conducted using these libraries and accompanying environments which would provide a great source for learning and experimenting with latest developments into generative music.
<https://www.tensorflow.org/> <https://magenta.tensorflow.org/>

- Logic Tensor Networks (LTN) Symbolic framework for Tensor Flow. Logic Tensor Networks by Garcez [12], has been implemented as a Python library to be used in Google TensorFlow. This library would be used with different neural architectures including RNNs, GRUs, LSTMs to represent musical domain knowledge as Real Logic, a variant of First Order Logic to impose music theoretic concepts and constraints towards the generative process.

3.3 Datasets

GTTM Database. Dataset to be used for this experiment comes with Hamanaka's GTTM visualizer [15]. This dataset is built upon music-theoretical analysis of 300 Western classical melodies and contains timespan and prolongation reduction PCFGs as parse trees of accompanied melodies in MusicXML format. These parse trees along the melodic data will be converted into NLP rules for processing within the models. <http://gttm.jp/gttm/database/>

Essen Folk Music Dataset. Essen folk music data set contains a collection of 6251 European folksongs. This dataset has been annotated with pitch, duration, metre information and phrase markers. This information makes the dataset a good candidate for musical sequence generation problem. The dataset, however does not contain any information on hierarchy and level of annotation, as compared to GTTM dataset is much lower. The dataset has been maintained by EsAC (Essen Associative Code) and can be found at <http://www.esac-data.org/>

Unlabelled Symbolic Music Datasets. Following unlabelled music datasets will also be used in unsupervised generation methods Piano midi dataset @ <http://www.piano-midi.de>, Nottingham Music Database @ <http://abc.sourceforge.net/NMD> and Muse Data @ <http://musedata.stanford.edu/>

3.4 Training Neural Networks

Neural networks, especially RNNs architectures with LSTMs and RBNs can easily learn, process and generate temporal and structural dependencies which makes them ideal candidates as generative music algorithms. A number of different models from selected literature will be implemented and tested with Tensor Flow and Magenta libraries and results will be compared to actual experiments. Selected neural computing algorithms will be trained on mentioned dataset with grammar rules to enforce the structural definitions on generated melodies while the ground truth provided with the dataset will be used for supervised learning

3.5 EVALUATION

A number of evaluation measures are being proposed to assess the model performance in terms of prediction accuracy, aesthetic appeal and conformance to music-theoretic principles as follows:

3.5.1 Turing Test Style Human Evaluation

Neural Symbolic AI based music generation systems are virtually able to create a near infinite amount of sounds and sequences when used with probabilistic rules. A "Turing Test" style evaluation of the system will be performed where experts and casual listeners will be asked to

try and distinguish between music created by humans and music generated by the proposed system as suggested by Miranda and Williams in [22]

3.5.2 Statistical Music Theoretic Measures

In order to quantitatively evaluate the generated monophonic melodies, further metrics motivated by music theory based rules will be used, as suggested in [28] for evaluating music generated with grammar augmented Neural Networks for music generation. Two such measures described in the referenced literature as:

- Percentage of notes in the diatonic scale $P(\text{dia})$. Most of western music is based upon diatonic scale made up of seven tones within one octave, which are used to generate various melodic sequences. $P(\text{dia})$ calculation will help identify un-necessary overtones and random/non-structured generated notes.
- Percentage of pitch intervals within one octave $P(\text{spi})$. This metric is based on idea that interval between two consecutive notes should not exceed an octave i.e. the Short Pitch Interval (spi). Large jumps between intervals may sound disturbing and un-natural. This metric will help us identify the presence of such notes within the generated sequences in order to determine the quality of generated music.

3.5.3 Cross Fold Validation

Supervised training and generation experiments will be evaluated through cross validation. A 5 – 7 fold cross fold evaluation is suggested in this case with averaged results. Cross fold validation loss will be considered as the primary measure of identifying prediction loss in the supervised training paradigm. In order to enhance consistency of proposed probabilistic model, same test and training data will be used for each cross-validation check. Visual and statistical outputs will be used as means of accuracy and loss as suggested by Groves in[14].

4 PROJECT PLAN

This section highlights some details on how the complete project will be broken down into smaller stages and sequence of activities with deliverables at each stage, risks and mitigation criteria.

4.1 Project Report

The report will begin to be formally drafted as soon as the model evaluation stage commences (as given in Gantt chart, Fig 3). This is because most of the information cannot finalised and put into text until some results have been obtained. The report will be finalised after two draft versions and reviews to reflect any changes or improvements in description of experiments conducted.

4.2 Project Stages

The project plan has been split into a number of stages as shown below:

- Stage 1 : Setting up Python programming environment with necessary libraries e.g. NLTK, Tensor Flow, Magenta, LTN and other as required.
- Stage 2 : The proposed data sets will be visually analysed and learnt with accompanying treebanks, grammar rules will be implemented and experimented with for a clear understanding into CFGs and probabilistic grammars.
- Stage 3 : Existing experiments with musical grammars and neural networks, as mentioned in references, will be replicated. Tensor Flow and Magenta libraries will be used for developing and evaluating multiple network architectures to identify the chosen model. LTN framework will be used for symbolic musical knowledge augmentation.
- Stage 4 : Selected model will be trained on grammatical rules applied on given dataset. Outputs will be carefully monitored and algorithms will be fine-tuned accordingly for improved accuracy and performance.
- Stage 5 : Final results of the chosen model will be evaluated using Turing style listening tests and music-theoretic measures as explained in evaluation section.
- Stage 6 : Results will be compiled and final report will be submitted on due date.

Figure 3 shows a risk register which highlights the main risks which are likely to be encountered in the proposed project while identifying the likelihood for each. The consequences that for each risk and total impact it may have on the project. A mitigation criterion is provided to manage these risks, should they occur during the project. These stages are graphically presented with information on deadlines, milestones and final deliverables as Gantt chart in Appendix A (Fig 4).

Description	Likelihood (1 – 3)	Consequence (1 – 5)	Impact (L x C)	Mitigation
Datasets are hard to understand/interpret and necessary information cannot be extracted	1	3	3	Look for alternative datasets, consult with the supervisor
Tensor Flow, LTN and other libraries are showing errors while installing and programming	2	3	6	Get expert advice, explore alternatives like Ubuntu Linux platform
NLTK libraries do not work with given dataset after conversion into ABC notation	2	4	8	Perform in-depth analysis of the conversion process, explore similar examples and consult the supervisor
Existing results cannot be replicated	1	3	3	Seek expert advice from supervisor and authors.
Neural Symbolic systems are hard to understand and work with.	2	5	10	Use example programs to understand the network architecture, seek advice from supervisor
Training Neural Networks with probabilistic grammar and given dataset takes too long to run.	2	3	6	Calculate the processing time required for the algorithm to run before-hand and plan the training phase accordingly, use simple models with experimentation
Generated code is lost or corrupted, hardware issues	1	5	5	Use backups, arrange a backup system.

Figure 3: . Risk Register with likelihood, consequence and impact with proposed action

References

- [1] Abdallah, S.A. and Gold, N.E., Comparing models of symbolic music using probabilistic grammars and probabilistic programming. 2014.
- [2] Baroni, M., Brunetti, R., Callegari, L. and Jacoboni, C. A grammar for melody: Relationships between melody and harmony Imperial Japan 1800-1945 Musical grammars and computer analysi 1984
- [3] Boulanger-Lewandowski, N., Bengio, Y. and Vincent, P., 2012. Modeling temporal dependencies in high- dimensional sequences: Application to polyphonic music generation and transcription.arXiv preprint arXiv:1206.6392.[4] Brown, S., J. Martinez, and Lawrence M. Parsons. (2006). Music and language side by side in the brain: a PET study of the generation of melodies and sentences. European Journal of Neuroscience, pages 2791–2803.
- [5] Cohen, W.W., 2016. TensorLog: A Differentiable Deductive Database.arXiv preprint arXiv:1605.06523.
- [6] Davismoon, S. and Eccles, J., 2010. Combining musical constraints with Markov transition probabilities to improve the generation of creative musical structures. Applications of Evolutionary Computation.

- [7] Diaz-Jerez, G., 2011. Composing with Melomics: Delving into the computational world for musical inspiration. *Leonardo Music Journal*, 21, pp.13-14.
- [8] Ebcioglu, K., 1988. An expert system for harmonizing four-part chorales. *Computer Music Journal*, 12(3), pp.43-51.
- [9] Eno, B., 1996, Generative music. *Imagination Conference*
- [10] Fernández, J.D. and Vico, F., 2013. AI methods in algorithmic composition: A comprehensive survey.. *Journal of Artificial Intelligence Research*, 48, pp.513-582.
- [11] Garcez, A.S.D.A., Broda, K. and Gabbay, D.M., 2012. *Neural-symbolic learning systems: foundations and applications*. Springer Science and Business Media.
- [12] Serafini, L. and Garcez, A.D.A., 2016. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. *arXiv preprint arXiv:1606.04422*.
- [13] Gilbert, É. And Conklin, D., 2007, January. A probabilistic context-free grammar for melodic reduction. *Proceedings of the International Workshop on Artificial Intelligence and Music, 20th International Joint Conference on Artificial Intelligence*. (pp. 83-94).
- [14] Groves, R., 2016. Automatic Melodic Reduction Using a Supervised Probabilistic Context-Free Grammar. *ISMIR* (pp. 775-781).
- [15] Hamanaka, M., Hirata, K. and Tojo, S., 2015, June. GTTM III: Learning-Based Time-Span Tree Generator Based on PCFG. *International Symposium on Computer Music Multidisciplinary Research*, (pp. 387-404). Springer International Publishing.
- [16] Hiller, L. and Isaacson, L.M., 1959. *Experimental Music. Composition with an Electronic Computer.* [With an Example in Musical Notation.]. McGraw-Hill Book Company.
- [17] Horner, A. and Goldberg, D.E., 1991. Genetic algorithms and computer-assisted music composition. *Urbana*, 51(61801), pp.437-441.
- [18] Hu, Z., Ma, X., Liu, Z., Hovy, E. and Xing, E., 2016. Harnessing deep neural networks with logic rules. *arXiv preprint arXiv:1603.06318*.
- [19] Jackendoff, R., 1985. *A generative theory of tonal music*. MIT press.
- [20] Jaques, N., Gu, S., Turner, R.E. and Eck, D., 2016. Tuning Recurrent Neural Networks with Reinforcement Learning. *arXiv preprint arXiv:1611.02796*
- [21] Loper, E. and Bird, S., 2002, July. NLTK: The natural language toolkit. *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics-Volume 1* (pp. 63-70). Association for Computational Linguistics.
- [22] Miranda, E.R. and Williams, D. (2015). *Artificial Intelligence in Organised Sound*. . *Organised Sound*, 20(1), pp.76–81.

- [23] Mozer, M.C., 1994. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, 6(2-3), pp.247-280.
- [24] Nayebi, A. and Vitelli, M., 2015. GRUV: Algorithmic Music Generation using Recurrent Neural Networks.
- [25] Nierhaus, G., 2009. *Algorithmic composition: paradigms of automated music generation*. Springer Science and Business Media.
- [26] Papadopoulos, G. and Wiggins, G., 1999, April. AI methods for algorithmic composition: A survey, a critical view and future prospects. *AISB Symposium on Musical Creativity*, (pp. 110-117). Edinburgh, UK.
- [27] Quick, D., 2015. Composing with kulitta. *ICMC*.
- [28] Sun, Z., Liu, J., Zhang, Z., Chen, J., Huo, Z., Lee, C.H. and Zhang, X., 2016. Grammar Argumented LSTM Neural Networks with Note-Level Encoding for Music Composition.
- [29] Thom, B., 2000, June. BoB: an interactive improvisational music companion. *Proceedings of the fourth international conference on Autonomous agents* (pp. 309-316). ACM.
- [30] Yang, L.C., Chou, S.Y. and Yang, Y.H., 2017. MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation using 1D and 2D Conditions. *arXiv preprint arXiv:1703.10847*.
- [31] Zicarelli, D., 1987. M and jam factory. *Computer Music Journal*, 11(4), pp.13-29.
- [32] Zimmermann, D., 1998, June. Modeling Musical Structures, Aims, Limitations and the Artist's Involvement. *Proc. Constraints techniques for artistic applications, Workshop at ECAI* (Vol. 98). Vancouve

Appendix 1

The proposal appendix contains a Gantt chart for project plan and an ethics form

This Gantt chart explains the proposed flow of activities and experiments for this research project. Further details will be added/ modified based on the performance and progress observed at initial stages of the project.

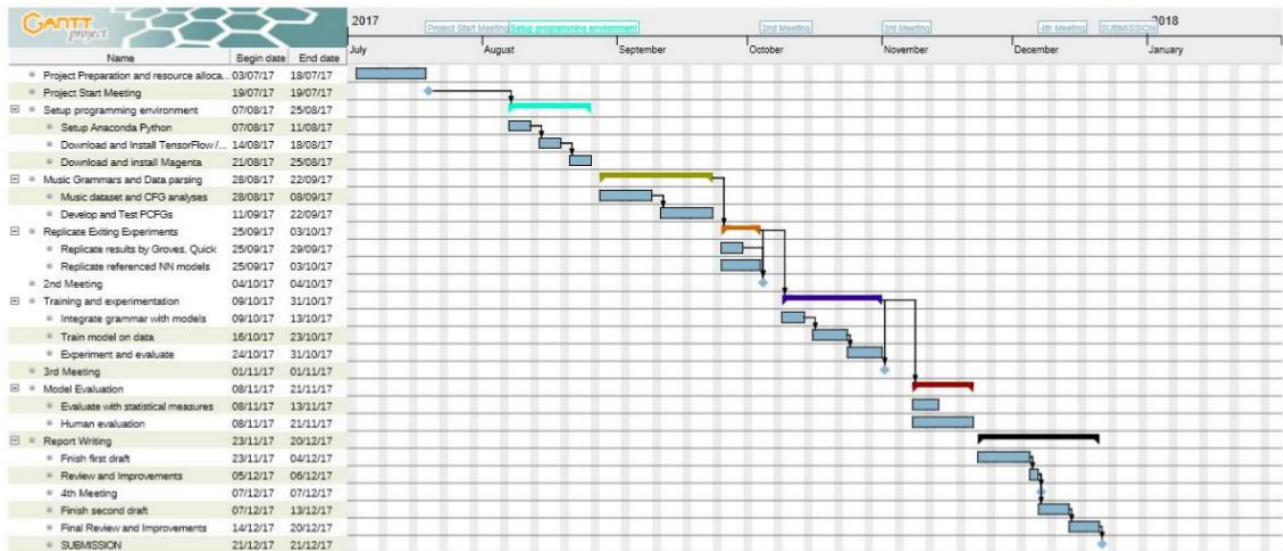


Figure 4: Gantt chart highlighting the flow and duration of proposed actions

Appendix 2.

Ethics Review Form: BSc, MSc and MA Projects

Computer Science Research Ethics Committee (CSREC)

Undergraduate and postgraduate students undertaking their final project in the Department of Computer Science are required to consider the ethics of their project work and to ensure that it complies with research ethics guidelines. In some cases, a project will need approval from an ethics committee before it can proceed. Usually, but not always, this will be because the student is involving other people ("participants") in the project.

In order to ensure that appropriate consideration is given to ethical issues, all students must complete this form and attach it to their project proposal document. There are two parts:

Part A: Ethics Checklist. All students must complete this part. The checklist identifies whether the project requires ethical approval and, if so, where to apply for approval.

Part B: Ethics Proportionate Review Form. Students who have answered "no" to questions 1 – 18 and "yes" to question 19 in the ethics checklist must complete this part. The project supervisor has delegated authority to provide approval in this case. The approval may be provisional: the student may need to seek additional approval from the supervisor as the project progresses.

A.1 If your answer to any of the following questions (1 – 3) is YES, you must apply to an appropriate external ethics committee for approval.		<i>Delete as appropriate</i>
1.	Does your project require approval from the National Research Ethics Service (NRES)? For example, because you are recruiting current NHS patients or staff? If you are unsure, please check at http://www.hra.nhs.uk/research-community/before-you-apply/determine-which-review-body-approvals-are-required/ .	No
2.	Does your project involve participants who are covered by the Mental Capacity Act? If so, you will need approval from an external ethics committee such as NRES or the Social Care Research Ethics Committee http://www.scie.org.uk/research/ethics-committee/ .	No
3.	Does your project involve participants who are currently under the auspices of the Criminal Justice System? For example, but not limited to, people on remand, prisoners and those on probation? If so, you will need approval from the ethics approval system of the National Offender Management Service.	No

A.2 If your answer to any of the following questions (4 – 11) is YES, you must apply to the City University Senate Research Ethics Committee (SREC) for approval (unless you are applying to an external ethics committee).		<i>Delete as appropriate</i>
4.	Does your project involve participants who are unable to give informed consent? For example, but not limited to, people who may have a degree of learning disability or mental health problem, that means they are unable to make an informed decision on their own behalf?	No
5.	Is there a risk that your project might lead to disclosures from participants concerning their involvement in illegal activities?	No

6.	Is there a risk that obscene and or illegal material may need to be accessed for your project (including online content and other material)?	No
7.	Does your project involve participants disclosing information about sensitive subjects? For example, but not limited to, health status, sexual behaviour, political behaviour, domestic violence.	No
8.	Does your project involve you travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning? (See http://www.fco.gov.uk/en/)	No
9.	Does your project involve physically invasive or intrusive procedures? For example, these may include, but are not limited to, electrical stimulation, heat, cold or bruising.	No
10.	Does your project involve animals?	No
11.	Does your project involve the administration of drugs, placebos or other substances to study participants?	No

A.3 If your answer to any of the following questions (12 – 18) is YES, you must submit a full application to the Computer Science Research Ethics Committee (CSREC) for approval (unless you are applying to an external ethics committee or the Senate Research Ethics Committee). Your application may be referred to the Senate Research Ethics Committee.		<i>Delete as appropriate</i>
12.	Does your project involve participants who are under the age of 18?	No
13.	Does your project involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)? This includes adults with cognitive and / or learning disabilities, adults with physical disabilities and older people.	No
14.	Does your project involve participants who are recruited because they are staff or students of City University London? For example, students studying on a specific course or module. (If yes, approval is also required from the Head of Department or Programme Director.)	No
15.	Does your project involve intentional deception of participants?	No
16.	Does your project involve participants taking part without their informed consent?	No
17.	Does your project pose a risk to participants or other individuals greater than that in normal working life?	No
18.	Does your project pose a risk to you, the researcher, greater than that in normal working life?	No

A.4 If your answer to the following question (19) is YES and your answer to all questions 1 – 18 is NO, you must complete part B of this form.		
19.	Does your project involve human participants or their identifiable personal data? For example, as interviewees, respondents to a survey or participants in testing.	Yes

Part B: Ethics Proportionate Review Form

If you answered YES to question 19 and NO to all questions 1 – 18, you may use this part of the form to submit an application for a proportionate ethics review of your project. Your project supervisor has delegated authority to review and approve this application. However, if you cannot provide all the required attachments (see B.3) with your project proposal (e.g. because you have not yet written the consent forms, interview schedules etc), the approval from your supervisor will be provisional. You **must** submit the missing items to your supervisor for approval prior to commencing these parts of your project. Failure to do so may result in you failing the project module.

There may also be circumstances in which your supervisor will ask you to submit a full ethics application to the CSREC, e.g. if your supervisor feels unable to approve your application or if you need an approval letter from the CSREC for an external organisation.

B.1 The following questions (20 – 24) must be answered fully.		<i>Delete as appropriate</i>
20.	Will you ensure that participants taking part in your project are fully informed about the purpose of the research?	Yes
21.	Will you ensure that participants taking part in your project are fully informed about the procedures affecting them or affecting any information collected about them, including information about how the data will be used, to whom it will be disclosed, and how long it will be kept?	Yes
22.	When people agree to participate in your project, will it be made clear to them that they may withdraw (i.e. not participate) at any time without any penalty?	Yes
23.	Will consent be obtained from the participants in your project? Consent from participants will be necessary if you plan to involve them in your project or if you plan to use identifiable personal data from existing records. “Identifiable personal data” means data relating to a living person who might be identifiable if the record includes their name, username, student id, DNA, fingerprint, address, etc. <i>If YES, you must attach drafts of the participant information sheet(s) and consent form(s) that you will use in section B.3 or, in the case of an existing dataset, provide details of how consent has been obtained. You must also retain the completed forms for subsequent inspection. Failure to provide the completed consent request forms will result in withdrawal of any earlier ethical approval of your project.</i>	Yes
24.	Have you made arrangements to ensure that material and/or private information obtained from or about the participating individuals will remain confidential? Provide details: A summarized version of data will be shown without highlighting individual preferences	Yes

B.2 If the answer to the following question (25) is YES, you must provide details		<i>Delete as appropriate</i>
25.	Will the research be conducted in the participant's home or other non-University location? <i>If YES, provide details of how your safety will be ensured:</i> It will be on-line survey.	Yes

B.3 Attachments (these should be provided if applicable):	<i>Delete as appropriate</i>
Participant information sheet(s)**	Yes
Consent form(s)**	Yes
Questionnaire(s)**	Yes
Topic guide(s) for interviews and focus groups**	Yes
Permission from external organisations (e.g. for recruitment of participants)**	Not applicable

If these items are not available at the time of submitting your project proposal, provisional approval through proportionate review can still be given, under the condition that you must submit the final versions of all items to your supervisor for approval at a later date. **All such items **must** be seen and approved by your supervisor before the activity for which they are needed starts.

APPENDIX B

PRE-PROCESSING AND FEATURE AUGMENTATION

This appendix contained extra images from the pre-processing stage.

B.1 Harmony and Melody Ranges selected for input encoding.

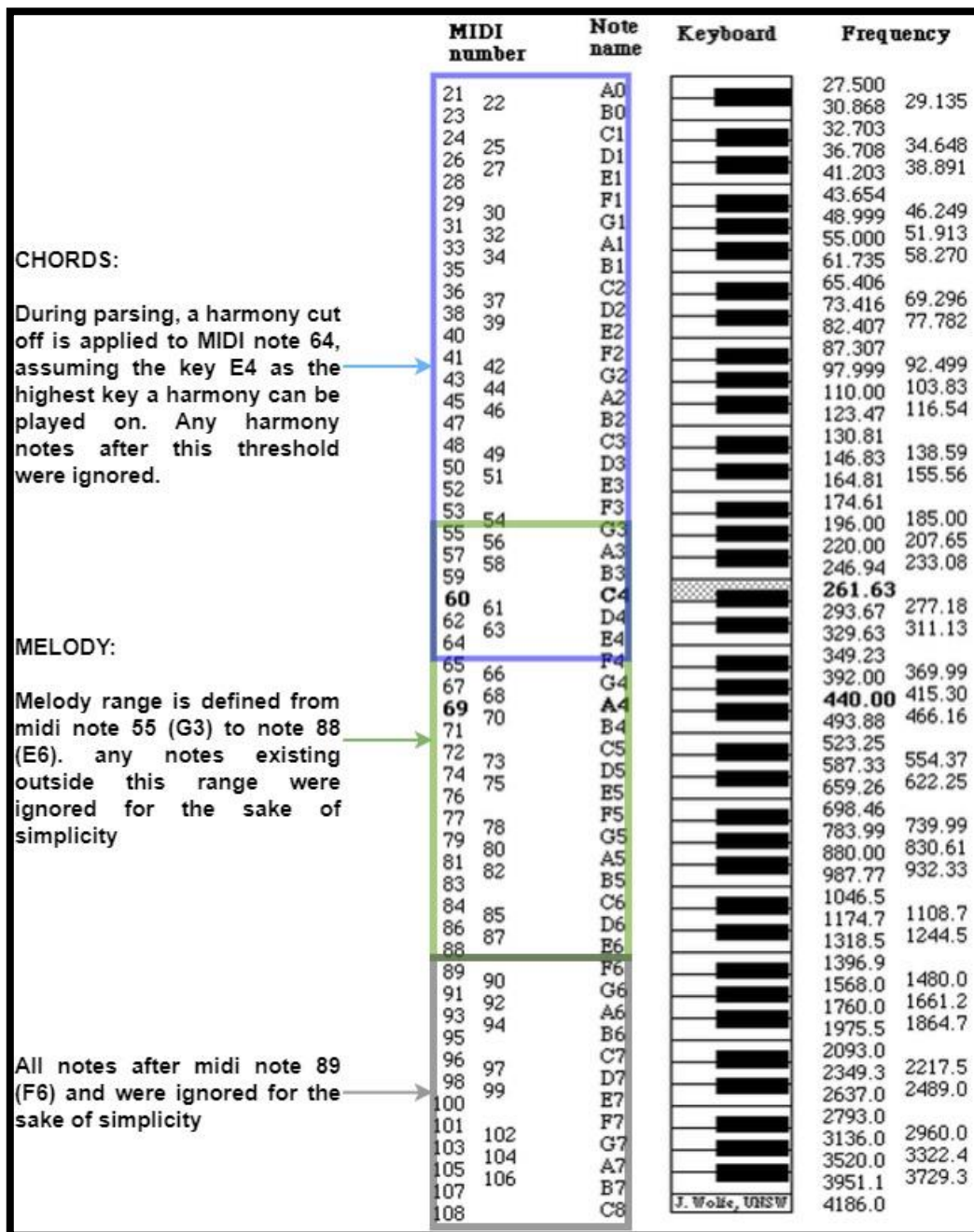


Fig B.1 This Figure highlights the melodic and harmonic note range defined for parsing midi files. The note range, defined by (Yoav, 2016) suits the structure of the files found within Nottingham music Dataset.

B.2 HARMONIC NOTES TO MINGUS PRE-PROCESSING

Following figure highlights the steps to change harmonic notes into chord classes as described in section 4.4.2.

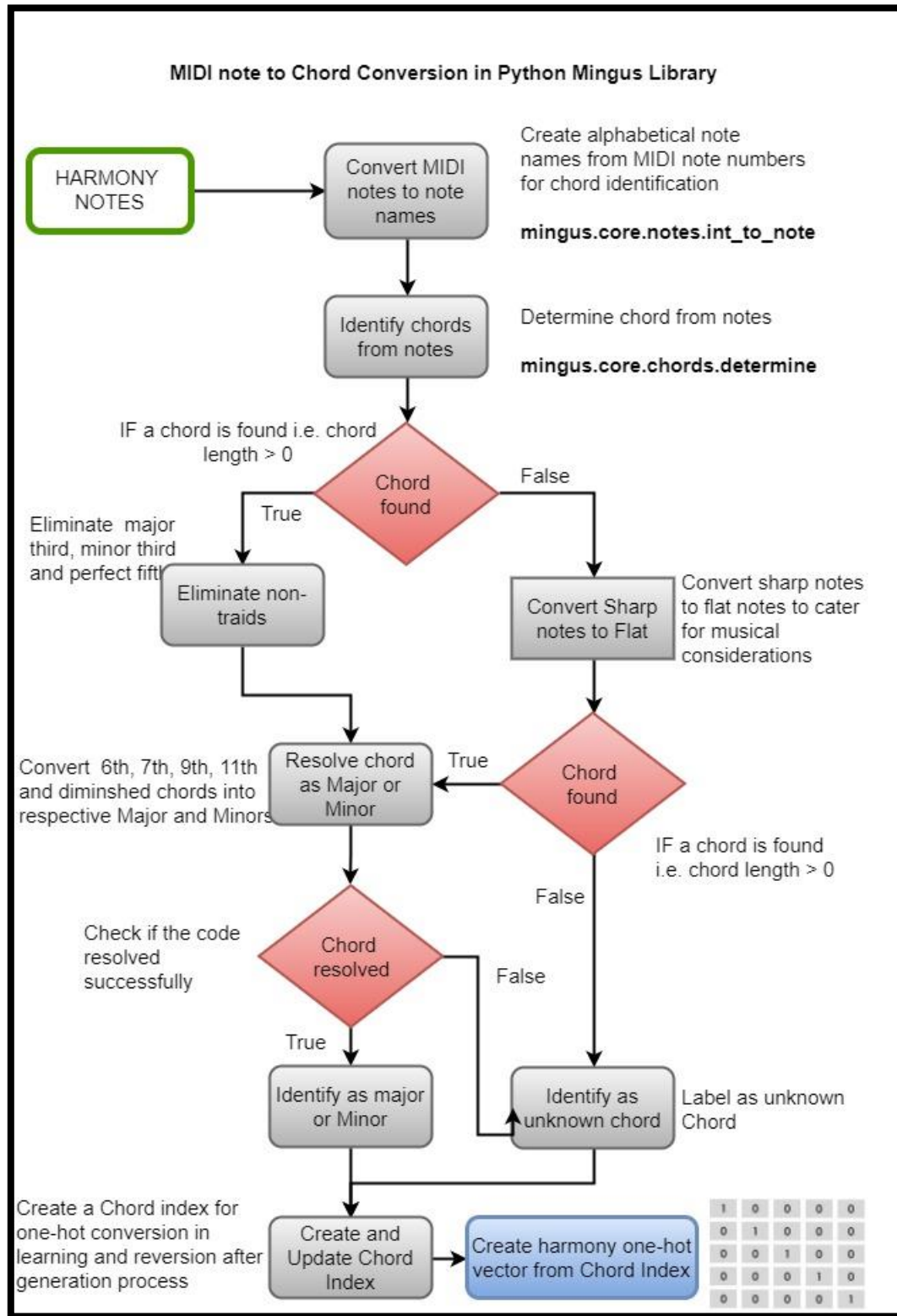


Fig B.2 Using Python Mingus library to Identify chord classes and to convert them into basic chords i.e. major and minor of each key.

APPENDIX C

Python Code for StructLSTM

This section provides the python script used for pre-processing, feature augmentation, model training and generation. The baseline model has been inspired by the work of (Yoav, 2016) and further additions to the code are performed towards the development of StructLSTM. This code with additional files is also being attached as a digital submission under the title **Appendix_C.zip** which also contains additional pre-processing files for midi parsing. A complete list of files in the digital submission include the following

- **preprocess.py (for pre-processing and feature augmentation)**
- **midi_util.py** (for MIDI files reading and writing)
- **train.py (for zero padding, mini-batching and training the model)**
- **model.py (model definition)**
- **composer_baseline.py** (for composing with baseline mode)
- **composer_counter_simple.py** (for composing with simple integer counter augmentation)
- **composer_counter_norm.py** (for composing with modified counter)
- **composer_structLSTM.py (for composing with durational as well as metrical features)**
- *data* folder with original Nottingham Dataset

The files presented here are those scripts which were modified and reflect the proposed architecture, and shown in bold above. **midi_util.py** includes script as used in the original code base and no modifications were made to this code. Extra **composer*** files contain intermediary generation scripts that were developed towards the StructLSTM model, and are not heavily commented. The attached files can also be viewed at the github repository at <https://github.com/ShakeelRaja/structlstm>

APPENDIX C.1

preprocess.py

```
1.  #!/usr/bin/env python2
2.  # -*- coding: utf-8 -*-
3.  """
4.  Shakeel ur Rehman Raja
5.  Final Project for MSc Data Science (2016/17). City, University of london
6.
7.
8.  StructLSTM - (Structure Augmented LSTM model)
9.
10. The code is inspired by the dual softmax model developed by Yoav Zimmerman,
11. at http://yoavz.com/music_rnn/. The baseline model has been developed by applying
12. a few midifications to the original code available at
13. https://github.com/yoavz/music_rnn
14.
15. This preprcessing script allows:
16.   Preprocessing MIDI files to crate input encoding for RNN
17.   Based on user preferences:
18.     Extracts structural information from the dataset and created engineered features.
19.     Augments the enginreered features to the input encoding
```

```

20.     saves the new dataset as a pickle file for training the network in train.py
21.
22. Features Augmentation can be performed by setting relevant flags in the main nbody of the program
23.
24.
25. **note** preprocessing, training and generation scripts are separate and must be run
26. individually
27. """
28. import numpy as np
29. import os
30. import midi
31. import cPickle
32. import copy
33. from pprint import pprint
34. import itertools
35. import midi_util
36. import mingus
37. import mingus.core.chords
38. import matplotlib.pyplot as plt
39.
40.
41. # array shifting for appending counterswith shifted values
42. def shift(length,num):
43.     return itertools.islice(itertools.cycle(length),num,num+len(length))
44.
45. # function to reduce all possible chords to either major and minor of each key
46. def Resolve_Chords(chord):
47.
48.     # teliminate chords played with two keys
49.     chordEliminate = ['major third', 'minor third', 'perfect fifth']
50.     if chord in chordEliminate:
51.         return None
52.     # take the first chord if two chords exist
53.     if "|" in chord:
54.         chord = chord.split("|")[0]
55.     # remove 7 , 11, 9 and 6th chord and replace with relevant major or minor
56.     if chord.endswith("7") or chord.endswith("9") or chord.endswith("6"):
57.         chord = chord[:-1]
58.     if chord.endswith("11"):
59.         chord = chord[:-2]
60.     # replace diminished chord with minor
61.     if chord.endswith("dim"):
62.         chord = chord[:-3] + "m"
63.     # add M to all Major chords
64.     if (not chord.endswith("m") and not chord.endswith("M")) or chord.endswith("#"):
65.         chord = chord + 'M'
66.
67.     return chord
68.
69.
70. def Data_to_Sequence(midFile, timeStep):
71.     unkChord = 'NONE' # for unknown chords
72.     harmony = []
73.     # get MIDI evet messages from each file
74.     midData = midi.read_midifile(midFile)
75.
76.     # store file path and name as meta info
77.     meta = {
78.         "path": midFile,
79.         "name": midFile.split("/")[-1].split(".")[0]
80.     }
81.     # check for length , 3 tracks for meta , melody and harmony
82.     if len(midData) != 3:
83.         return (meta, None)
84.
85.     for msg in midData[0]:
86.         # get meta information
87.         if isinstance(msg, midi.TimeSignatureEvent):
88.

```

```

89.     # get PPQN (Pulse per Quarter Note)for MIDI resolution
90.     meta["ticks_per_quarter_note"] = msg.get_metronome()
91.     # get time signature values
92.     num = midData[0][2].data[0]
93.     dem = 2**(midData[0][2].data[1])
94.     sig = (num, dem)
95.     meta["signature"] = sig
96.
97.     # Measure time signatur frequency
98.     if sig not in sigs:
99.         sigs[sig] = 1
100.    else:
101.        sigs[sig] += 1
102.
103.    # Filter out sequences with time signature 4/4 based on flag
104.    if fourByFour == True:
105.        if (num == 3 or num == 6) or (dem !=4):
106.            return (meta, None)
107.
108.    # Track ingestion
109.    nTicks = 0
110.
111.    # get melody and harmony notes and ticks from midi Data
112.    melNotes, melTicks = midi_util.ingest_notes(midData[1])
113.    harNotes, harTicks = midi_util.ingest_notes(midData[2])
114.
115.    # round number of ticks with given time step value
116.    nTicks = midi_util.round_tick(max(melTicks, harTicks), timeStep)
117.
118.    # get melody encodings mapped to defined melody range
119.    melSequence = midi_util.round_notes(melNotes, nTicks, timeStep, R=melodyRange, O=melodyMin)
120.
121.    # filter out sequences with a double note is found in melody
122.    for i in range(melSequence.shape[0]):
123.        if np.count_nonzero(melSequence[i, :]) > 1:
124.            return (meta, None)
125.
126.    # get harmony sequence and process with Mingus
127.    harSequence = midi_util.round_notes(harNotes, nTicks, timeStep)
128.
129.    # convert sharps to flats to consider compositional considerations
130.    flat_note = {"A#": "Bb", "B#": "C", "C#": "Db", "D#": "Eb", "E#": "F", "F#": "Gb", "G#": "Ab",}
131.
132.    # use Mingus to identify chords from harmony notes
133.    for i in range(harSequence.shape[0]):
134.
135.        # get note data
136.        notes = np.where(harSequence[i] == 1)[0]
137.        if len(notes) > 0:
138.
139.            # get note names without octave information
140.            noteName = [ mingus.core.notes.int_to_note(note%12) for note in notes]
141.            chordName = mingus.core.chords.determine(noteName, shorthand=True)
142.
143.            if len(chordName) == 0:
144.                # convert to flat if chord not identified and try again
145.                noteName = [ flat_note[n] if n in flat_note else n for n in noteName]
146.                chordName = mingus.core.chords.determine(noteName, shorthand=True)
147.
148.            if len(chordName) == 0:
149.                # if chord does not exist, label as NONE
150.                if len(harmony) > 0:
151.                    harmony.append(harmony[-1])
152.                else:
153.                    harmony.append(unkChord)
154.
155.            # resolve chords as major or minor for other types of chord
156.            else:
157.                resolvedChord = Resolve_Chords(chordName[0])

```

```

158.         if resolvedChord:
159.             harmony.append(resolvedChord)
160.         else:
161.             harmony.append(unkChord)
162.     else:
163.         # label as unresolved/unknown
164.         harmony.append(unkChord)
165.
166.     return (meta, (melSequence, harmony))
167.
168. def Parse_Data(directory, timeStep):
169.     #returns a list of [T x D] matrices, where each matrix represents a
170.     #a sequence with T time steps over D dimensions
171.     midFiles = [ os.path.join(directory, d) for d in os.listdir(directory)
172.                 if os.path.isfile(os.path.join(directory, d))]
173.
174.     # retrieve melody and harmony sequences from files
175.     midiSequences = [Data_to_Sequence(f, timeStep=timeStep) for f in midFiles ]
176.
177.     # filter out the sequence if 2 tracks of melody and harmony are not found
178.     midiSequences = filter(lambda x: x[1] != None, midiSequences)
179.
180.     return midiSequences
181.
182.
183. def combine(mel, har):
184.
185.     unkChord = 'NONE' # for unknown chords
186.     final = np.zeros((mel.shape[0], melodyRange + numChords))
187.
188.     # for all melody sequences that don't have any notes, add the empty melody marker (last one)
189.     for i in range(mel.shape[0]):
190.         if np.count_nonzero(mel[i, :]) == 0:
191.             mel[i, melodyRange - 1] = 1
192.         # add melodies to final array
193.         final[:, :mel.shape[1]] += mel
194.
195.         # store chords on NONE if not found
196.         harIdx = [ chordMap[x] if x in chordMap else chordMap[unkChord] for x in har ]
197.         harIdx = [ melodyRange + y for y in harIdx ]
198.
199.         # return combined melody and harony one hot vector encoding
200.         # final vector will have exactly two 1's ( for melody and hamrmony)
201.         final[np.arange(len(har)), harIdx] = 1
202.
203.     return final
204.
205. """
206. PreProcessing for structure augmented LSTMs
207.
208. The program uses following flags to create different datasets depending
209. on the experiment. Setting a certain flag to True would enable the code
210. to augment the selected features into the dataset it creates after pre-processing
211.
212. """
213. zeroPad = True    # Zero pad data
214. counter = True    # include duration counter
215. normCount = True  # Normalize counter values
216. Shift = True      # shift conclusion to beginning of last note
217.
218. counterQB = False # count 4 time steps leading to one quarter beat (not used in experiments due to poor results)
219. counterB = True   # count cquarter beats leading to a whole bar of music
220. fourByFour = True # Select only 4/4 sequences for processing
221.
222. # Define original data and Saved data location
223.
224. # For final experiment
225.
226. dataLoc = 'data/Nottingham/{}'

```



```

227.pickleLoc= 'data/nottingham_duration_metrical_44only.pickle'
228.
229.
230.#For practicing, comment for actual experiments
231.#pickleLoc = 'data/nottingham_subset.pickle'
232.#data_loc = 'data/Nottingham_subset/{}'
233.
234. """
235.Program variables definition and initialization
236. """
237.
238.# Define Range of melody to be used based on MIDI note numbers
239.melodyMax = 88 # E6
240.melodyMin = 55 # G3
241.melodyRange = melodyMax - melodyMin + 1 + 1 # +1 for rest
242.
243.# Highest midi note used for detecting harmonies
244.chordLimit=64 # E4
245.
246.# Initilize arrays and dictionaries for pre-processing
247.sigs = {} # time signatures
248.data = {} # storing data prior to combining
249.final = {} # final dataset
250.chords = {} # store chords from Mingus
251.sequenceLength = [] # length of sequences
252.sequenceMax = 0 # maximum sequence length , 0 be default
253.sequenceMin = 1000 # minimum sequence length
254.timeStep = 120 # time step chosen for parsing data files
255.
256. """
257.Main pre-processing section
258. """
259.
260.if __name__ == "__main__":
261.
262. # Splitting and Parsing MIDI files
263.
264. for type in ["train", "valid"]:
265.     print "Parsing {} data".format(type)
266.     print "-----"
267.
268.     # get parsed data from dataset
269.     parsedData = Parse_Data(dataLoc.format(type), timeStep)
270.     # meta information from index 0
271.     meta = [m[0] for m in parsedData]
272.     # melodic and harmonic sequences at index 1
273.     seqPased = [s[1] for s in parsedData]
274.     data[type] = seqPased # store harmony and melody vectors
275.     data[type + '_meta'] = meta # store meta information
276.
277.     #store sequence lengths for analysis
278.     seqLengths = [len(seq[1]) for seq in seqPased]
279.
280.     #print seqLengths
281.     print "Maximum sequence length found: {}".format(max(seqLengths))
282.     print "Minimum sequence length found: {}".format(min(seqLengths))
283.     print ""
284.     sequenceLength += seqLengths # for calculating average sequence length
285.     sequenceMax = max(sequenceMax, max(seqLengths))
286.     sequenceMin = min(sequenceMin, min(seqLengths))
287.
288.
289.# Following code will extract extra features based on the duration of song in
290.# timesteps, normalising the counter values ad shifting the counter back , based
291.# on selected flags.
292.
293.## COUNTER + NORMALIZATION
294.
295. # create counter vector , reverse and store in the data dictionary for appending later

```

```

296.     if counter == True:
297.         length = []
298.         rev = []
299.         shifts = []
300.         length += [range(x) for x in seqLengths] # calculate song length
301.         for y in length:
302.             y = y[::-1] # reverse the counter as "countDOWN"
303.             # normalize counter values
304.             if normCount == True:
305.                 y2 = copy.deepcopy(y)
306.                 norm = [float(i)/max(y) for i in y] # optional normalize
307.                 y = norm
308.                 tt = [[item] for item in y]
309.                 rev += [tt] # append reversed counter
310.             # append counter to final data
311.             data[type + '_count'] = rev
312.
313. ## SHIFTED COUNTER
314.
315.     # Calculate the value of shifts required to move 0 back to beginning of last note
316.     if Shift == True:
317.         # deep copy reverse array for shifting
318.         rev2 = copy.deepcopy(rev)
319.         shifted = []
320.         # swap current and previous until note changes
321.         for x, y in seqPased:
322.             n = -1
323.             now = x[n,:]
324.             prev = x[n-1,:]
325.
326.             # step back from the end of song to find the beginning of last note
327.             # to find the number of shifts required
328.             while (now == prev).all() :
329.                 n = n-1
330.                 now = x[n,:]
331.                 prev = x[n-1,:]
332.                 shifts += [abs(n+1)]
333.
334.             # roll back counter to point the zero at beginning of last note
335.             for count, roll in zip(rev2, shifts):
336.                 [count.append([0]) for x in range(roll)]
337.                 count = list(shift(count, roll))
338.                 count = count[: -roll]
339.                 shifted += [count]
340.
341.             # append shifted counter values to final data
342.             data[type + '_count_shift'] = shifted
343.
344. ## TIMESTEP COUNTER TO ONE QUARTER BEAT
345.
346.     # adding quarterbeat/4 timestep information
347.     if counterQB == True:
348.         # create the quarter beat counter one hot vector
349.         tSteps = [[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]]
350.         # tSteps = [[1], [2], [3], [4]]
351.         tStepFinal = []
352.
353.         # duration, repetition of quarterbeats and remaining timesteps
354.         for dur in seqLengths:
355.             rep = dur//4 # number of quarterbeats in the sequence (for 4/4)
356.             rem = dur%4 # extra number of quarter beats (considering 4/4)
357.             tStepVec = []
358.             for _ in range(rep):
359.                 tStepVec += tSteps
360.             if rem > 0:
361.                 padStep = []
362.                 for _ in range(rem):
363.                     # Calculate length of padding of 0 for pre-song notes
364.                     padStep += [[0,0,0,0]]

```



```

365.         #padStep += [[0]]
366.         tStepFinal += [padStep+ tStepVec]
367.     else:
368.         tStepFinal += ([tStepVec])
369.
370.     # append quarter beat vector to final dataset
371.     data[type + '_countQB'] = tStepFinal
372.
373. ## TIMESTEP COUNTER TO FULL BEAT in 4/4
374.
375.     # calculating and adding full beat one hot vector
376.     if counterB == True:
377.         # create one hot vectors
378.         tSteps = [[1,0,0,0], [1,0,0,0], [1,0,0,0], [1,0,0,0],
379.                   [0,1,0,0], [0,1,0,0], [0,1,0,0], [0,1,0,0],
380.                   [0,0,1,0], [0,0,1,0], [0,0,1,0], [0,0,1,0],
381.                   [0,0,0,1], [0,0,0,1], [0,0,0,1], [0,0,0,1]]
382.         tStepFinal = []
383.         for dur in seqLengths:
384.
385.             rep = dur//4
386.             rem = dur%4
387.             # duration , repitition of quarterbeats and remianing timesteps
388.             #print dur, rep, rem
389.             tStepVec = []
390.             x = rem + 1
391.             y = 0
392.             # cycle through tSteps and calculate the relevant one hot vectora
393.             # at each timestep
394.             while x <= dur:
395.                 tStepVec += [tSteps[y]]
396.                 y = 0 if y == 15 else y+1
397.                 x = x+1
398.             # add extra padding
399.             if rem > 0:
400.                 padStep = []
401.                 for _ in range(rem):
402.                     padStep += [[0,0,0,0]]
403.                 tStepFinal += [padStep+ tStepVec]
404.             else:
405.                 tStepFinal += ([tStepVec])
406.
407.         # append full beat vector to final dataset
408.         data[type + '_countB'] = tStepFinal
409.
410.     # count frequencies of chord occurances in the dataset
411.     for _, harmonySeq in seqPased:
412.         for c in harmonySeq:
413.             if c not in chords:
414.                 chords[c] = 1
415.             else:
416.                 chords[c] += 1
417.
418. #Calculate average length, which may be used for identifying batch timestep length
419. sequenceAvg = float(sum(sequenceLength)) / len(sequenceLength)
420.
421. #Prepare chord index for harmony one hot vector
422. chordLimit=64
423. # calculate chords and frequencies
424. chords = { chord: ind for chord, ind in chords.iteritems() if chords[chord] >= chordLimit }
425. chordMap = { chord: ind for ind, chord in enumerate(chords.keys()) }
426. numChords = len(chordMap)
427.
428. # append chord index to final
429. final['chordIx'] = chordMap
430.
431. #plot the chord distribution chart
432. pprint(chords)
433. plt.figure(figsize=(10, 4))

```

```

434. plt.bar(range(len(chords)), chords.values())
435. plt.xticks(range(len(chords)), chords.keys())
436. plt.show()
437.
438. #print sequence information
439. print "Total sequences parsed: {}".format(len(sequenceLength))
440. print "Maximum length of sequences: {}".format(sequenceMax)
441. print "Minimum length of sequences: {}".format(sequenceMin)
442. print "Average length of sequences: {}".format(sequenceAvg)
443. print "Number of chords found: {}".format(numChords)
444.
445. # combine sequences with counters based on set flags
446. def attach(data, counter):
447.     result = []
448.     result = [np.hstack((a[0], np.array(b[0])))]
449.     for i in range(1, len(a)):
450.         result.append(np.hstack((a[i], np.array(b[i]))))
451.     final[item] = result
452.     return None
453.
454.
455. #Combine melody and harmony vectors
456. for item in ["train", "valid"]:
457.     print "Combining {}".format(item)
458.     #combine melody and harmony one hot vectors into a single vector
459.     final[item] = [ combine(mel, har) for mel, har in data[item] ]
460.     final[item + '_meta'] = data[item + '_meta']
461.
462.
463. #save pickle data with optional counters
464. if counter == True:
465.     a = final[item]
466.     if Shift == True:
467.         b = data[item + '_count_shift']
468.     else:
469.         b = data[item + '_count']
470.     attach(a,b)
471. if counterQB == True:
472.     a = final[item]
473.     b = data[item + '_countQB']
474.     attach(a,b)
475. if counterB == True:
476.     a = final[item]
477.     b = data[item + '_countB']
478.     attach(a,b)
479.
480. # save final dataset to pickle location
481. filename=pickleLoc
482. with open(filename, 'w') as f:
483.     cPickle.dump(final, f, protocol=-1)

```

APPENDIX C.2

train.py

```
1.  #!/usr/bin/env python2
2.  # -*- coding: utf-8 -*-
3.  """
4.  Shakeel ur Rehman Raja
5.  Final Project for MSc Data Science (2016/17). City, University of london
6.
7.  StructLSTM - (Structure Augmented LSTM model)
8.
9.  The code is inspired by the dual softmax model developed by Yoav Zimmerman,
10. at http://yoavz.com/music\_rnn/. The baseline model has been developed by applying
11. a few midifications to the original code available at
12. https://github.com/yoavz/music\_rnn
13.
14. This code performs following function on datasets created in preprocess.py
15.     Loads a dataset indicated by preprocess.py
16.     Performs zero-padding on the dataset for preserving sequence duration
17.     Performs mini-batching on data and creates features and labels dataset
18.     Removes augmented features from labels and makes it compatible with Language Modelling
19.     Uses examples and labels for training and validation of the model
20.     Runs a parametric grid search with defined ranges
21.     saves the best models and training output automatically during training
22.
23. Setting the zero-pad flag in preprocess.py allows the code
24. to add a padding to make it a multiple of 128 (mini-batch length).
25.
26. The code creates a config file with model settings for later use within composer scripts
27.
28.
29. **note** preprocessing, training and generation scripts are separate and must be run
30. individually
31.
32. """
33.
34. import os, sys
35. import argparse
36. import time
37. import itertools
38. import cPickle
39. import logging
40. import random
41. import string
42. import preprocess
43.
44. from collections import defaultdict
45. from random import shuffle
46.
47.
48. import numpy as np
49. import tensorflow as tf
50. import matplotlib.pyplot as plt
51.
52. from model import Model, NottinghamModel
53.
54. def configName(config):
55.
56.     # print configuration settings on console
57.     def replace_dot(s): return s.replace(".", "p")
58.     return " num_layers_" + str(config.num_layers) + \
59.         "_hidden_size_" + str(config.hidden_size) + \
60.         replace_dot("_melCoef_{}".format(config.melody_coeff)) + \
61.         replace_dot("_dropout_prob_{}".format(config.dropout_prob)) + \
62.         replace_dot("_inpdropout_prob_{}".format(config.input_dropout_prob)) + \
63.         replace_dot("_timeBatchLen_{}".format(config.time_batch_len))+ \
64.         replace_dot("_cellType_{}".format(config.cell_type))
65.
66. # This class defines the default parameters for training the model, settings in
```

```

67. # grid search will overwrite these values.
68. class DefaultConfig(object):
69.
70.     # default model achitecture
71.     num_layers = 2
72.     hidden_size = 200
73.     melody_coeff = 0.5
74.     dropout_prob = 0.5
75.     input_dropout_prob = 1    # set to 1 for no dropout_prob at input layer
76.     cell_type = 'LSTM'    # use "GRU" for GRUs and "Vanilla" for simple RNN
77.
78.     # default learning parameters
79.     max_time_batches= 10
80.     time_batch_len= 128
81.     learning_rate = 1e-3
82.     learning_rate_decay = 0.9
83.     num_epochs = 250
84.
85.     # metadata
86.     dataset = 'softmax'
87.     modelFile = ""
88.
89.
90.     def __repr__(self):
91.         return """Num Layers: {}, Hidden Size: {}, dropout_prob Prob: {}, Cell Type: {}, Time Batch Len: {}, Learning Rate:
           {}, Decay: {}""".format(self.num_layers, self.hidden_size, self.dropout_prob, self.cell_type, self.time_batch_len, self.learning_rate, self.learning_rate_decay)
92.     data = []
93.     targets = []
94.
95.     # Runs training and validation with model and resturns loss values to main program
96.     def run_epoch(session, model, batches, training=False, testing=False):
97.
98.         # shuffle batches
99.         shuffle(batches)
100.
101.         # set target tensors for testing and training
102.         target_tensors = [model.loss, model.final_state]
103.         if testing:
104.             target_tensors.append(model.probs)
105.             batch_probs = defaultdict(list)
106.         if training:
107.             target_tensors.append(model.trainStep)
108.
109.         losses = []
110.         for data, targets in batches:
111.             # save state over unrolling time steps
112.             batch_size = data[0].shape[1]
113.             nTimeSteps = len(data)
114.             state = model.get_cell_zero_state(session, batch_size)
115.             probs = list()
116.
117.             for tb_data, tb_targets in zip(data, targets):
118.                 if testing:
119.                     tbd = tb_data
120.                     tbt = tb_targets
121.                 else:
122.                     # shuffle all the batches of input, state, and target *FOR TRAINING ONLY*
123.                     batches = tb_data.shape[1]
124.                     permutations = np.random.permutation(batches)
125.                     tbd = np.zeros_like(tb_data)
126.                     tbd[:, np.arange(batches), :] = tb_data[:, permutations, :]
127.                     tbt = np.zeros_like(tb_targets)
128.                     tbt[:, np.arange(batches), :] = tb_targets[:, permutations, :]
129.                     state[np.arange(batches)] = state[permutations]
130.
131.             # prepare input features and labels for model training
132.             feed_dict = {
133.                 model.initial_state: state,

```

```

134.     model.inSeq: tbd,
135.     model.targetSeq: tbt,
136. }
137. results = session.run(target_tensors, feed_dict=feed_dict)
138.
139. #save losses and state for training next mini-batch
140. losses.append(results[0])
141. state = results[1]
142. if testing:
143.     batch_probs[nTimeSteps].append(results[2])
144.
145. loss = sum(losses) / len(losses)
146.
147. # return training and validation losses
148. if testing:
149.     return [loss, batch_probs]
150. else:
151.     return loss
152.
153.
154. def Batch_Data(seqIn, nTimeSteps):
155.
156.     seq = [s[: (nTimeSteps*time_batch_len)+1, :] for s in seqIn]
157.
158.     # stack sequences depth wise (along third axis).
159.     stacked = np.dstack(seq)
160.     # swap axes so that shape is (SEQ_LENGTH X BATCH_SIZE X inputDim)
161.     data = np.swapaxes(stacked, 1, 2)
162.     # roll data -1 along length of sequence for next sequence prediction
163.     targets = np.roll(data, -1, axis=0)
164.     # cutoff final time step, cut of count from targets
165.     data = data[:-1, :, :]
166.     targets = targets[:-1, :, :] #-1 in 3rd dimension to eliminate counter from targets
167.
168. # assert data.shape == targets.shape #works without counter
169.
170. labels = np.ones((targets.shape[0], targets.shape[1], 2), dtype=np.int32)
171. #create melody and harmony labels
172. labels[:, :, 0] = np.argmax(targets[:, :, :preprocess.melodyRange], axis=2)
173. labels[:, :, 1] = np.argmax(targets[:, :, :preprocess.melodyRange:], axis=2)
174. targets = labels
175.
176. # ensure data and target integrity
177. assert targets.shape[:2] == data.shape[:2]
178. assert data.shape[0] == nTimeSteps * time_batch_len
179.
180. # split sequences into time batches
181. tb_data = np.split(data, nTimeSteps, axis=0)
182. tb_targets = np.split(targets, nTimeSteps, axis=0)
183.
184. return (tb_data, tb_targets)
185.
186. def pause():
187.     programPause = raw_input("Press the <ENTER> key to continue...")
188.
189. """
190. Main program code
191. """
192.
193. np.random.seed(0)
194. softmax = True
195.
196. loc = preprocess.pickleLoc #location of dataset created in preprocessing
197. counter = preprocess.counter # counter flag
198. modelLoc = 'models'
199. runName = time.strftime("%m%d_%H%M")
200. time_step = 120
201. modelClass = NottinghamModel
202.

```

```

203. # read data saved in preprocess.py
204. with open(loc, 'r') as f:
205.     pickle = cPickle.load(f)
206.     chordIx = pickle['chordIx']
207.
208. inputDim = pickle["train"][0].shape[1] #+1 for counter
209. print 'Data loaded, with total number of input dimension = {}'.format(inputDim)
210.
211. # set up run dir for saving training data
212. runLoc = os.path.join(modelLoc, runName)
213. if os.path.exists(runLoc):
214.     raise Exception("{} already exists, select a different folder".format(runLoc))
215. os.makedirs(runLoc)
216.
217. #start logger for training and validation runs
218. logger = logging.getLogger(__name__)
219. logger.handlers = []
220. logger.setLevel(logging.INFO)
221. logger.addHandler(logging.StreamHandler())
222. logger.addHandler(logging.FileHandler(os.path.join(runLoc, "training.log")))
223.
224. #setup a grid for trying combinations of hyperparameters
225. # ranges can be defined as [value1,value2...valuen]
226. paramGrid = {
227.     "dropout_prob": [0.8],
228.     "melody_coeff": [0.5],
229.     "num_layers": [2],
230.     "hidden_size": [150],
231.     "num_epochs": [50],
232.     "learning_rate": [5e-3],
233.     "learning_rate_decay": [0.9],
234. }
235.
236. # Generate a combination of hyper parameters defined in the grid
237. runs = list(list(itertools.izip(paramGrid, x)) for x in itertools.product(*paramGrid.itervalues()))
238. logger.info("a total of {} runs will be performed".format(len(runs)))
239.
240. # intitlize training variables
241. bestConfig = None
242. bestValidLoss = None
243. time_batch_len = 128
244. comb = 1
245.
246. #Run the combinations identified from grid parameters
247. for combination in runs:
248.     #load grid values to config
249.     config = DefaultConfig()
250.     config.dataset = 'softmax'
251.     #create model with random name
252.     config.model_name = ".join(random.choice(string.ascii_uppercase + string.digits) for _ in range(12)) + '.model'
253.     for attr, value in combination:
254.         setattr(config, attr, value)
255.
256.     dataSplit = {}
257.     for dataset in ['train', 'valid']:
258.         # For testing, use ALL the sequences
259.         # for counter, use same length of testing and training to allow retention of augmented features
260.         #if counter == False:
261.         if dataset == 'valid':
262.             max_time_batches = -1
263.         else:
264.             max_time_batches = 10
265.
266.         # load data
267.         sequences = pickle[dataset]
268.         metadata = pickle[dataset + '_meta']
269.         dims = sequences[0].shape[1]
270.         seqLens = [s.shape[0] for s in sequences]
271.

```

```

272.     avgSeqLen = sum(seqLens) / len(sequences)
273.
274.     # print basic information about the dataset
275.     if comb == 1:
276.         print "Dataset: {}".format(dataset)
277.         print "Ave. Sequence Length: {}".format(avgSeqLen)
278.         print "Max Sequence Length: {}".format(time_batch_len)
279.         print "Number of sequences: {}".format(len(sequences))
280.         print "_____ "
281.
282.     batches = defaultdict(list)
283. #
284. # for zero padding, comment out for truncating sequences
285. # creates padding for input data to make it a multiple of mini-batch length
286. for sequence in sequences:
287.     if (sequence.shape[0]-1) % time_batch_len == 0 :
288.         nTimeSteps = ((sequence.shape[0]-1) // time_batch_len)
289.     else:
290.         #calculate the pad size and create new sequence
291.         nTimeSteps = ((sequence.shape[0]-1) // time_batch_len) + 1
292.         pad = np.zeros((nTimeSteps*time_batch_len+1, sequence.shape[1]))
293.         pad[:,sequence.shape[0]:sequence.shape[1]] = sequence
294.         sequence = pad
295.
296.     if nTimeSteps < 1:
297.         continue
298.     if max_time_batches > 0 and nTimeSteps > max_time_batches:
299.         continue
300.
301. # for truncating sequences, comment out for zero padding and feature augmentation
302. # for sequence in sequences:
303. #     # -1 because we can't predict the first step
304. #     nTimeSteps = ((sequence.shape[0]-1) // miniBatchLen)
305. #     if nTimeSteps < 1:
306. #         continue
307. #     if max_time_batches > 0 and nTimeSteps > max_time_batches:
308. #         continue
309.
310.     batches[nTimeSteps].append(sequence)
311.
312.     # create batches of examples based on sequence length/minibatches
313.     batchedData = [Batch_Data(bSeq, tStep) for tStep, bSeq in batches.iteritems()]
314.
315.     #add metadata to batched data (just in case)
316.     dataSplit[dataset] = {
317.         "data": batchedData,
318.         "metadata": metadata,
319.     }
320.     dataSplit["inputDim"] = batchedData[0][0][0].shape[2]
321.
322. # save number of input dimensions to configuration file
323. config.input_dim = dataSplit["inputDim"]
324.
325. print "
326. print 'Combination no. {}'.format(comb)
327. print "
328. logger.info(config)
329. configPath = os.path.join(runLoc, configName(config) + 'RUN_'+str(comb)+'_'+ '.config')
330. with open(configPath, 'w') as f:
331.     cPickle.dump(config, f)
332. #
333. #%%
334. # build tensorflow models for training and validation from model.py
335. with tf.Graph().as_default(), tf.Session() as session:
336.     with tf.variable_scope("model", reuse=None):
337.         train_model = modelClass(config, training=True)
338.     with tf.variable_scope("model", reuse=True):
339.         valid_model = modelClass(config, training=False)
340.

```

```

341. saver = tf.train.Saver(tf.all_variables(), max_to_keep=40)
342. tf.initialize_all_variables().run()
343.
344. # training variables initialization
345. earlyStopBestLoss = None
346. modelSave = False
347. saveFlag = False
348. tLosses, vLosses = [], []
349. start_time = time.time()
350.
351. # train and validate the model calculating loss at each iteration
352. for i in range(config.num_epochs):
353.     loss = run_epoch(session, train_model,
354.         dataSplit["train"]["data"], training=True, testing=False)
355.     tLosses.append((i, loss))
356.     if i == 0:
357.         continue
358.     # perform validation on trained model
359.     vLoss = run_epoch(session, valid_model, dataSplit["valid"]["data"], training=False, testing=False)
360.     vLosses.append((i, vLoss))
361.     logger.info('Epoch: {}, Train Loss: {}, Valid Loss: {}, Time Per Epoch: {}'.format(\
362.         i, loss, vLoss, (time.time() - start_time)/i))
363.
364.     # save current model if new validation loss goes higher or lower than current best validation loss
365.     if earlyStopBestLoss == None:
366.         earlyStopBestLoss = vLoss
367.     elif vLoss < earlyStopBestLoss:
368.         earlyStopBestLoss = vLoss
369.         if modelSave:
370.             logger.info('Best model seen. model saved')
371.             saver.save(session, os.path.join(runLoc, config.model_name))
372.             saveFlag = True
373.         elif not modelSave:
374.             modelSave = True
375.             logger.info('Valid loss increased, previous model was saved')
376.             saver.save(session, os.path.join(runLoc, config.model_name))
377.             saveFlag = True
378.
379.     # save model if not saved already
380.     if not saveFlag:
381.         saver.save(session, os.path.join(runLoc, config.model_name))
382. #
383. # plot train and validation loss curves
384. axes = plt.gca()
385. axes.set_ylim([0, 3])
386.
387. # Save the loss values as a png file in the model folder
388. plt.plot([t[0] for t in tLosses], [t[1] for t in tLosses])
389. plt.plot([t[0] for t in vLosses], [t[1] for t in vLosses])
390. plt.legend(['Train Loss', 'Validation Loss'])
391. chart_file_path = os.path.join(runLoc, configName(config) + '_RUN_' + str(comb) + '_' + '.png')
392. plt.savefig(chart_file_path)
393. plt.clf()
394.
395. # log the best model and config
396. logger.info("Config {}, Loss: {}".format(config, earlyStopBestLoss))
397. if bestValidLoss == None or earlyStopBestLoss < bestValidLoss:
398.     logger.info("Found best new model!")
399.     bestValidLoss = earlyStopBestLoss
400.     bestConfig = config
401.
402. comb = comb+1
403.
404. # identify best loss when training ends
405. logger.info("Best Config: {}, Loss: {}".format(bestConfig, bestValidLoss))

```


APPENDIX C.3

model.py

```
1.  #!/usr/bin/env python2
2.  # -*- coding: utf-8 -*-
3.  """
4.  Shakeel ur Rehman Raja
5.  Final Project for MSc Data Science (2016/17). City, University of london
6.
7.  StructLSTM - (Structure Augmented LSTM model)
8.
9.  The code is inspired by the dual softmax model developed by Yoav Zimmerman,
10. at http://yoavz.com/music\_rnn/. The baseline model has been developed by applying
11. a few midfications to the original code available at
12. https://github.com/yoavz/music\_rnn
13.
14. This code performs following function on batches of features and labels
15.   Initilazes a Tensorflow RNN model based on configuration file
16.   Identifies input and output dimension based on augmented features (needs manual adjustment)
17.   Performs the Dual Softmax calculation suggested by (Yoav, 2016)
18.   Calculates the training and validation loss for each mini-batch and returns values to training script
19.   ** Note ** the model architecture uses melody-co-efficient of 0.5 to give equal weight to melody and
20.       harmony loss
21.
22. The code creates a config file with model settings for later use within composer scripts
23.
24.
25. **note** preprocessing, training and generation scripts are separate and must be run
26. individually
27.
28.
29. """
30. import tensorflow as tf
31. from tensorflow.models.rnn import rnn_cell
32. from tensorflow.models.rnn import rnn, seq2seq
33. import numpy as np
34.
35. import preprocess
36.
37. class Model(object):
38.     """
39.     inSeq: a [ T x B x D ] matrix, where T is the time steps in the batch, B is the
40.         batch size, and D is the amount of dimensions.
41.     """
42.     # initialize model parameters from config file
43.     def __init__(self, config, training=False):
44.         self.config = config
45.         self.miniBatchLen = miniBatchLen = config.miniBatchLen
46.         self.inputDim = inputDim = config.inputDim
47.         hidSize = config.hidSize
48.         numLayers = config.numLayers
49.         dropOut = config.dropOut
50.         inDropOut = 1
51.         rnnCell = config.rnnCell
52.
53.         #create a placeholder for input sequence
54.         self.inSeq = tf.placeholder(tf.float32, shape=[self.miniBatchLen, None, inputDim])
55.
56.         # reduce features from the output (change this to 0 = no augmentation, 1 = counter only
57.         # and 5 for counter + metrical augmentation)
58.         outDim= self.inputDim
59.
60.         # setup variables
61.         with tf.variable_scope("RNN"):
62.             outWeight = tf.get_variable("outWeight", [hidSize, outDim])
63.             outBias = tf.get_variable("outBias", [outDim])
64.             self.lr = tf.constant(config.learnRate, name="learning_rate")
```

```

65.         self.lrDecay = tf.constant(config.learnRateDecay, name="learning_rate_decay")
66.
67.         # create RNN cell type based on definition in config file
68.         # set as LSTM by default and can be changed in train.py to GRU or Simple RNN
69.         def create_cell(input_size):
70.             if rnnCell == "Vanilla":
71.                 cell_class = rnn_cell.BasicRNNCell
72.             elif rnnCell == "GRU":
73.                 cell_class = rnn_cell.GRUCell
74.             elif rnnCell == "LSTM":
75.                 cell_class = rnn_cell.BasicLSTMCell
76.             else:
77.                 raise Exception("Invalid cell type: {}".format(rnnCell))
78.             cell = cell_class(hidSize, input_size = input_size)
79.
80.         #apply output dropout to training data
81.         if training:
82.             return rnn_cell.DropoutWrapper(cell, output_keep_prob = dropOut)
83.         else:
84.             return cell
85.
86.         #create input sequence applying dropout to training data
87.         # input drop out has been set to 1 i.e. no dropout for this experiment
88.         if training:
89.             self.inDropOut = tf.nn.dropout(self.inSeq, keep_prob = inDropOut)
90.         else:
91.             self.inDropOut = self.inSeq
92.
93.         # create an n layer (num_layer) sized MultiRnnCell, defining sizes for each
94.         self.cell = rnn_cell.MultiRNNCell([create_cell(inputDim)] + [create_cell(hidSize) for i in range(1, numLayers)])
95.
96.         # batch size = number of timesteps i.e. 128 , initial 0 state and input+dropout tensor
97.         batchSize = tf.shape(self.inDropOut)[0]
98.         self.initial_state = self.cell.zero_state(batchSize, tf.float32)
99.         inputs = tf.unpack(self.inDropOut)
100.
101.         # rnn outputs a list of [batchSize x H] outputs
102.         outputs, self.final_state = rnn.rnn(self.cell, inputs, initial_state=self.initial_state)
103.
104.         # get the outputs, calculate output activations
105.         outputs = tf.pack(outputs)
106.         opConcat = tf.reshape(outputs, [-1, hidSize])
107.         logitConcat = tf.matmul(opConcat, outWeight) + outBias
108.
109.         #Reshape output tensor
110.         logits = tf.reshape(logitConcat, [self.miniBatchLen, -1, outDim])
111.
112.         # probabilities of each note
113.         self.probs = self.calculate_probs(logits)
114.         self.loss = self.init_loss(logits, logitConcat)
115.         self.trainStep = tf.train.RMSPropOptimizer(self.lr, decay = self.lrDecay).minimize(self.loss)
116.
117.         # loss calculation and note probabilities without dual softmax
118.         def init_loss(self, outputs, _):
119.             self.targetSeq = tf.placeholder(tf.float32, [self.miniBatchLen, None, self.inputDim])
120.
121.             batchSize = tf.shape(self.inSeq_dropout)
122.             crossEntropy = tf.nn.sigmoid_cross_entropy_with_logits(outputs, self.targetSeq)
123.             return tf.reduce_sum(crossEntropy) / self.miniBatchLen / tf.to_float(batchSize)
124.
125.         def calculate_probs(self, logits):
126.             return tf.sigmoid(logits)
127.
128.         # initiliaze the model with cell states
129.         def get_cell_zero_state(self, session, batchSize):
130.             return self.cell.zero_state(batchSize, tf.float32).eval(session=session)
131.
132.     class NottinghamModel(Model):
133.         """

```

```

134. Dual softmax formulation as described by (Yoav, 2016)
135. Applied to dataset with structure augmented features allowing melodies and
136. harmonies to learn from augmentation.
137.
138. """
139.
140. #initiliaze with model
141. def init_loss(self, outputs, opConcat):
142.     self.targetSeq = tf.placeholder(tf.int64, [self.miniBatchLen, None, 2])
143.     batchSize = tf.shape(self.targetSeq)[1]
144.
145.     with tf.variable_scope("RNN"):
146.         self.melCoeff = tf.constant(self.config.melCoeff)
147.
148.         targets_concat = tf.reshape(self.targetSeq, [-1, 2])
149.
150.         #Dual softmax calculates individual melody and harmony losses
151.         melLoss = tf.nn.sparse_softmax_cross_entropy_with_logits( \
152.             opConcat[:, :preprocess.melodyRange], \
153.             targets_concat[:, 0])
154.         harLoss = tf.nn.sparse_softmax_cross_entropy_with_logits( \
155.             opConcat[:, preprocess.melodyRange:], \
156.             targets_concat[:, 1])
157.         losses = tf.add(self.melCoeff * melLoss, (1 - self.melCoeff) * harLoss)
158.         return tf.reduce_sum(losses) / self.miniBatchLen / tf.to_float(batchSize)
159.
160. # apply softmax to melody and harmony output and append the probabiliy values
161. def calculate_probs(self, logits):
162.     steps = []
163.     for timeStep in range(self.miniBatchLen):
164.         softmaxMelody = tf.nn.softmax(logits[timeStep, :, :preprocess.melodyRange])
165.         softmaxHarmony = tf.nn.softmax(logits[timeStep, :, preprocess.melodyRange:])
166.         steps.append(tf.concat(1, [softmaxMelody, softmaxHarmony]))
167.     return tf.pack(steps)
168.
169. def assign_melCoeff(self, session, melCoeff):
170.     if melCoeff < 0.0 or melCoeff > 1.0:
171.         raise Exception("Invalid melody coeffecient")
172.
173.     session.run(tf.assign(self.melCoeff, melCoeff))

```

APPENDIX C.4

structLSTM.py

```
1.  """
2.  Shakeel ur Rehman Raja
3.  Final Project for MSc Data Science (2016/17). City, University of london
4.
5.  StructLSTM - (Structure Augmented LSTM model)
6.
7.  The code is inspired by the dual softmax model developed by Yoav Zimmerman,
8.  at http://yoavz.com/music\_rnn/. The baseline model has been developed by applying
9.  a few midifications to the original code available at
10. https://github.com/yoavz/music\_rnn
11.
12. This code performs following function on datasets created in preprocess.py
13.   Loads a dataset indicated by preprocess.py
14.   Performs zero-padding on the dataset for preserving sequence duration
15.   Performs mini-batching on data and creates features and labels dataset
16.   Removes augmented features from labels and makes it compatible with Language Modelling
17.   Uses examples and labels for training and validation of the model
18.   Runs a parametric grid search with defined ranges
19.   saves the best models and training output automatically during training
20.
21. Setting the zero-pad flag in preprocess.py allows the code
22. to add a padding to make it a multiple of 128 (mini-batch length).
23.
24. The code creates a config file with model settings for later use within composer scripts
25.
26.
27. **note** preprocessing, training and generation scripts are separate and must be run
28. individually
29.
30. """
31. import os
32. import argparse
33. import cPickle
34.
35. import numpy as np
36. import tensorflow as tf
37. import copy
38.
39. import midi_util
40. import preprocess
41. from model import Model, NottinghamModel
42. from rnn2 import DefaultConfig
43.
44. # library to generate a chord sequence for conditioning the model
45. def i_vi_iv_v(chord_to_idx, repeats, input_dim):
46.     r = preprocess.Melody_Range
47.     input_dim = input_dim - 5
48.
49.     i = np.zeros(input_dim)
50.     i[r + chord_to_idx['CM']] = 1
51.     vi = np.zeros(input_dim)
52.     vi[r + chord_to_idx['Am']] = 1
53.     iv = np.zeros(input_dim)
54.     iv[r + chord_to_idx['FM']] = 1
55.     v = np.zeros(input_dim)
56.     v[r + chord_to_idx['GM']] = 1
57.
58.     full_seq = [i] * 16 + [vi] * 16 + [iv] * 16 + [v] * 16
59.     full_seq = full_seq * repeats
60.
61.     return full_seq
62. """
```

```

63. Uncomment one of the following to use required dataset
64. """
65. # Use 4/4 subset of data with duration and metrical augmented features (improved generation)
66. file = '/home/shaks/Desktop/Music_RNN/data/nottingham_allin_notStep_44.pickle'
67. config_file = 'models/44_no_Ts/numLayers_2_hidSize_200_melCoef_0p5_dropOut_0p5_inpDropOut_1_timeBatchLen
    _128_cellType_LSTMRUN_5_config'
68.
69. # Use complete dataset with duration and metrical augmented features (produces low quality results due to mixed s
    ignature)
70. #file = 'data/nottingham_allin_notStep.pickle'
71. #config_file = 'models/all_no_Ts/numLayers_2_hidSize_200_melCoef_0p5_dropOut_0p3_inpDropOut_1_timeBatchLe
    n_128_cellType_LSTMRUN_4_config'
72.
73. conditioning=32
74. sampleLength=384
75. # Use random for randomly selected conditioning sequence from validation data, or use chords (doesnt work well m
    etrical augmentation)
76. sample_seq = 'random' #choices = ['random', 'chords'])
77. time_step = 120 # midi resolution and set timestep for savin data
78. resolution = 480
79.
80.
81. # main program
82. if __name__ == '__main__':
83.
84.     np.random.seed()
85.
86.     # open saved config file for loading trained model
87.     with open(config_file, 'r') as f:
88.         config = cPickle.load(f)
89.         if config.dataset == 'softmax':
90.             config.time_batch_len = 1
91.             config.max_time_batches = -1
92.             model_class = NottinghamModel
93.             with open(file, 'r') as f:
94.                 pickle = cPickle.load(f)
95.                 chord_to_idx = pickle['chord_to_idx']
96.             print (config)
97. #%%
98. # create the sampling model with config settings
99. with tf.Graph().as_default(), tf.Session() as session:
100.     with tf.variable_scope("model", reuse=None):
101.         sampling_model = model_class(config)
102.
103.     saver = tf.train.Saver(tf.all_variables())
104.     model_path = os.path.join(os.path.dirname( config_file),
105.         config.model_name)
106.     saver.restore(session, model_path)
107.     state = sampling_model.get_cell_zero_state(session, 1)
108.
109.     # Create condition sequence based on user choice
110.     if sample_seq == 'chords':
111.         # 16 - one measure, 64 - chord progression
112.         repeats = sampleLength / 64
113.         sample_seq = i_vi_iv_v(chord_to_idx, repeats, config.input_dim)
114.         print ('Sampling melody using a I, VI, IV, V progression')
115.
116.     elif sample_seq == 'random':
117.         sample_index = np.random.choice(np.arange(len(pickle['valid'])))
118.         sample_seq = [ pickle['valid'][sample_index][i, :-5]
119.             for i in range(pickle['valid'][sample_index].shape[0]) ]
120.
121.     # create Feature Generator for augmenting structural features
122.     lenVec = []
123.     rev = []
124.     length = sampleLength
125.     lenVec = [range(length)]
126.     for y in lenVec:
127.         y = y[::-1]

```

```

128.     y2 = copy.deepcopy(y)
129.     norm = [float(i)/max(y) for i in y] # optional normalize
130.     y = norm
131.     tt = ([[item] for item in y])
132.     rev +=[tt]
133.
134.     # Append the normalised counter to the conditioning sequences
135.     ptr = 0
136.     chord1 = sample_seq[0]
137.     seq = [chord1]
138.     chord = np.append(chord1, rev[0][ptr])
139.
140.     # create array for metrical features augmentation
141.     mSteps = [[1,0,0,0], [1,0,0,0], [1,0,0,0], [1,0,0,0],
142.               [0,1,0,0], [0,1,0,0], [0,1,0,0], [0,1,0,0],
143.               [0,0,1,0], [0,0,1,0], [0,0,1,0], [0,0,1,0],
144.               [0,0,0,1], [0,0,0,1], [0,0,0,1], [0,0,0,1]]
145.     mCount = 0
146.     chord = np.append(chord, mSteps[mCount])
147.
148.     # condition the model with feature augment sequences for steps defined
149.     if conditioning > 0:
150.         ptr = ptr + 1
151.         for i in range(1, conditioning):
152.             seq_input = np.reshape(chord, [1, 1, config.input_dim])
153.             feed = {
154.                 sampling_model.seq_input: seq_input,
155.                 sampling_model.initial_state: state,
156.             }
157.             state = session.run(sampling_model.final_state, feed_dict=feed)
158.             chord = sample_seq[i]
159.             seq.append(chord)
160.
161.     # Augment and append features index for next time step
162.     if rev[0][ptr] == 0:
163.         ptr = ptr
164.     else:
165.         ptr = ptr + 1
166.         chord = np.hstack((chord, rev[0][ptr]))#
167.         if mCount < 15:
168.             mCount += 1
169.         elif mCount == 15:
170.             mCount = 0
171.         chord = np.hstack((chord, mSteps[mCount]))
172. # # initialize sampling and MIDI saving
173.
174. writer = midi_util.NottinghamMidiWriter(chord_to_idx, verbose=False)
175. sampler = midi_util.NottinghamSampler(chord_to_idx, verbose=False)
176.
177. probb = [] #for viewing probabilities
178.
179. # start the generation and sampling
180. for i in range(max(sampleLength - len(seq)+ 1, 0)):
181.     seq_input = np.reshape(chord, [1, 1, config.input_dim])
182.
183.     feed = {
184.         sampling_model.seq_input: seq_input,
185.         sampling_model.initial_state: state,
186.     }
187.     [probs, state] = session.run(
188.         [sampling_model.probs, sampling_model.final_state],
189.         feed_dict=feed)
190.     #Calculate probabilities through sampling
191.     probs = np.reshape(probs, [config.input_dim-5])
192.     probb.append(probs)
193.     chordx = sampler.sample_notes(probs)
194.
195.     # augment features for next sampling cycle
196.     chord = np.append(chordx, rev[0][ptr])

```

```

197.     x = rev[0][ptr]
198.     if float(x[0]) > 0:
199.         ptr = ptr + 1
200.         chord = np.hstack((chord, mSteps[mCount]))
201.         if mCount < 15:
202.             mCount += 1
203.         elif mCount == 15:
204.             mCount = 0
205.         seq.append(chordx)
206.
207.     # repeat the last note for 1 bar to avoid abrupt ending
208.     seq = seq[:-2]
209.     for i in range(16):
210.         seq.append(seq[-1])
211.
212.     # print probabilities of generated sequences
213.     prob_matrix = np.array(probb)
214.     import matplotlib.pyplot as plt
215.     plt.imshow(prob_matrix.T, cmap = "Reds", interpolation='nearest')
216.     plt.show()
217.     # Save probabilities as csv
218.     np.savetxt("models/zzGENERATED/all_noTS/probs_s3.csv", prob_matrix, delimiter=",")
219.     # Write output to MIDI File
220.     writer.dump_sequence_to_midi(seq, "models/zzGENERATED/all_noTS/all_noTS_s3.mid",
221.         time_step=time_step, resolution=resolution)

```

APPENDIX D

GENERATED PRODUCTS

Following products are being submitted in digital format along with this report, these products can also be accessed at the GitHub repository for this project at:

<https://github.com/ShakeelRaja/structlstm/tree/master/Appendices>

D.1 FEATURE AUGMENTED DATASETS

This contains datasets developed with different feature augmentation approaches. A number of different datasets were tested and tried. Digital appendix titled ***Appendix_D1_datasets.zip*** is a zip file containing following datasets presented in line with project objectives and experimented with in this report.

- **baseline.pickle:** baseline model without feature augmentation for comparison
- **simple_counter.pickle:** Simple counter augmentation in integer format
- **normalised_shifted_counter.pickle:** Normalisation and shift of conclusion to the beginning of last note.
- **normalised_shifted_counter_metrical_allData.pickle:** Duration counter and metrical feature augmentation to complete Nottingham dataset
- **normalised_shifted_counter_metrical_4by4Data.pickle:** Duration counter and metrical feature augmentation to 4/4 music from Nottingham dataset

D.2 TRAINED MODELS

The models trained on datasets above are being presented as appendix D.2. In the digital submission. File ***Appendix_D2_Trained_Models.zip*** contains best models from the grid searches shown in the results section. Following models are included in their own directories within this zip file

- baseline
- simple_counter
- normalised_shifted_counter
- normalised_shifted_counter_metrical_allData
- normalised_shifted_counter_metrical_4by4Data

APPENDIX E

SUBJECTIVE EVALUATION AND ETHICAL REQUIREMENTS

This appendix contains documents and information used and generated within the subjective evaluation phase. Detailed user responses (**Appendix E.4**) and data extracted from these tests (**Appendix E.5**) is being attached as digital appendix due to length and nature of these documents.

E.1 PARTICIPANT INFORMATION:

We would like to invite you to take part in a research study. Before you decide whether you would like to take part it is important that you understand why the research is being done and what it would involve for you. Please take time to read the following information carefully and discuss it with others if you wish. Ask us if there is anything that is not clear or if you would like more information.

The primary research question, that this project attempts to answer is:

"Can we improve the performance of a Recurrent Deep Neural Network architecture by augmenting structural knowledge using a Language Modelling framework, in order to stochastically generate musical sequences while enforcing structural and metrical constraints on the generative process"

You have been invited to participate in this study in the capacity of listener/ musicians/ professionals or an enthusiast. This survey has been designed to get your feedback on the music generated by the model mentioned above. Your participation in this survey is voluntary, and you can choose not to participate in part or all of the project. You can withdraw at any stage of the project without being penalized or disadvantaged in any way. It is up to you to decide whether or not to take part. If you do decide to take part, you will be asked to sign a consent form. If you decide to take part, you are still free to withdraw at any time and without giving a reason.

The survey will take about 10 - 15 minutes of your time where you will be required to listen to a few musical sequences (including human performances and computer-generated melodies). You will then be asked to answer a few questions about each musical sequence you have listened. The information provided by you will primarily be used to evaluate the performance of the model in a Turing style evaluation.

There are no potential disadvantages for you if you take part in this study. Your participation, however, will contribute towards the developments in the field of Artificial Intelligence, Neural computing and music informatics. We do not ask any personal information from you except your previous musical knowledge and your initials to satisfy the consent and ethics requirements. At the end of the study you will be notified with the results.

Thank you for your time and effort.
Shakeel Raja

E.2 CONSENT FORM and USER INFORMATION

Kindly read following information and click on "I agree" to proceed.

I confirm that I have had the project explained to me, and I have read the participant information sheet, which I may keep for my records. I understand this will involve listening to given music samples and answering questions about them. I understand that any information I provide is confidential, and that no information that could lead to the identification of any individual will be disclosed in any reports on the project, or to any other party. No identifiable personal data will be published. The identifiable data will not be shared with any other organization. I understand that my participation is voluntary, that I can choose not to participate in part or all of the project, and that I can withdraw at any stage of the project without being penalized or disadvantaged in any way. I agree to City, University of London recording and processing this information about me. I understand that this information will be used only for the purpose(s) set out in this statement and my consent is conditional on City complying with its duties and obligations under the Data Protection Act 1998. I agree to City, University of London recording and processing this information about me. I understand that this information will be used only for the purpose(s) set out in this statement and my consent is conditional on City complying with its duties and obligations under the Data Protection Act 1998.

I agree to take part in the above study.

☒ I agree

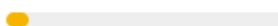
Previous familiarity with musical concepts *

- ☐ Not interested in music
- ☐ Casual listener , appreciate good music but not familiar with musical concepts
- ☒ Understand music theory and/or sing/play a musical instrument
- ☐ Music professional / enthusiast

Kindly Provide your Name/initials *

Joe Bloggs|_____

NEXT

 Page 1 of 12

E.3 INTRODUCTION

These tests will require you to listen to a mix of different human and machine composed musical sequences. You are requested to listen to these samples with following considerations and give a score for each question, for all sequences. Kindly answer all questions and give a mean score of 3 if you remain indecisive.

1. Likelihood of the sequence generated by a human vs. an intelligent computational algorithm. You are required to give a score based on your previous experience of listening to different kinds of music. Human compositions are normally identified as having a "soul" or a deeper emotional impact on the listener while computer generated music, although can sound musically correct, they might lack in the emotional dimension.
2. Long term musical structure: The long-term structure (or global structure) of a musical composition is identified by the similarity between different parts of the song and how these parts are repeated within the composition. The long-term structure is usually understood in terms of intro, chorus, verse, bridge and outro. While giving a score for long term structure, you are required to listen for similar grouping of bars, repeating within the music. If a part gets repeated, it may be a sign that the composition is bound by a global or a long-term structure.
3. Short term musical structure: The short-term structure of a musical composition is defined by the combinations and repetition of notes within adjacent beats and bars. The rhythm, metre (time signature) and key of the composition usually define the short-term structure of musical patterns within a composition. You are required to look for combinations of notes (phrases), their short-term repetitions holding the rhythm of the music and how well this rhythm is maintained.
4. How well does the song conclude? You are required to focus on the conclusion (ending) of the composition and give your opinion, based on your past listening experience if the song has concluded in a good manner, or did it end abruptly.
5. 5. How would you score overall quality of this musical sequence? You are required to give each sequence a score based on how pleasant the sequence sounded to you.