Automatic Harmonic Analysis of Melodies

Finlay McAfee

MInf Project (Part 2) Report
Master of Informatics
School of Informatics
University of Edinburgh

2017

Abstract

This report proposes two models for the automatic harmonic analysis of melodies. One is a generative Hidden Markov Model, the other is a discriminative Long Short Term Memory Neural Network. Each model deals only with symbolic input, notes in the form of 12 possible pitch classes, and outputs a sequence of predicted chord roots. The models are trained on two datasets, the Weimar Jazz Database (WJazz) of jazz solo transcriptions and the KP corpus of classical melody excerpts. The models perform well on the KP corpus when compared to two proposed baselines: a model which chooses the tonic chord at every times step and a model which chooses the chord with the root the same as the first note in the input. The models do not show any improvement on the *Tonic* model for the jazz data, confirming the suspicion of last years report, that this dataset is not suitable to the task.

Acknowledgements

I would like to thank Mark Steedman and Andrew McLeod for their guidance on this project, Adam Lopez for organising group meetings and giving helpful advice, and Salma Saade and Sandy McAfee for making sure that this report makes sense to humans.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Finlay McAfee*)

Table of Contents

Chapter 1

Introduction

When listening to a melody, most people will get a sense of it moving through some kind of space separate from the notes it is comprised of, a harmonic space. Perhaps it begins joyfully and becomes melancholic. It will create tension and then release it. It embodies some kind of higher structure than just the sequences of notes themselves.

This harmonic structure that we intuitively hear in melodies is directly related to any chord progression that may be accompanying the melody. In a sense, that chord progression exists behind the melody even if it is not being played. This implies a direct relationship between the notes and their accompanying chords.

This relationship is exploited in the generation of melodies for models that take a sequence of chords as input and generate melodies from them, see [4]. In theory then, one should be able to generate a sequence of chords from an observed melody. In this instance the problem looks more like classification; we can think of the chord sequence as 'generating' the observed notes, and we are trying to determine the most likely *hidden* chord sequence.

This report is concerned with the question of whether that chord sequence can be determined from the symbolic notes themselves, without any extra audio or beat related information, which is where the majority of the literature has been focussed, see section 2.5 for an overview.

Two models will be presented in this report on the prediction of chord sequences from note sequences, one based on Hidden Markov Models (HMM) and one based on Long-Short-Term-Memory (LSTM) Neural Networks.

1.1 Previous Year

In the previous year of this project, work was started on the HMM implementation, with the focus on the relationship between a single chord and the notes that were associated with it. Three models of this relationship were proposed, one based on a naive frequency distribution (Bag of Notes), one based on a complex decision tree structure

(Chord Tone Model) and one sequential model that implemented a second, lower level HMM (Concatenative HMM).

These models were implemented on transcriptions of jazz solos and compared to an implementation of a non-statistical template matching model from [52].

The conclusion of the year's work was that the data was unsuitable for the task, with all models, including the Template model, not being able to outperform a baseline of choosing the tonic chord repeatedly. However the Chord Tone Model and the Concatenative HMM outperformed the Template model so this suggested that this approach was worth persuing.

1.2 Summary of contributions

The contributions of this year's work are summarized below for reference:

- Before any new developments could be made it was necessary to search for a new dataset, as the results of the previous year's work showed that the data being used was insufficient for training a purely symbolic note to chord model. The dataset that was found was the KP Corpus of classical melody excerpts [32].
- This new dataset was in a different format from the original data, hence a new data processing pipeline had to be implemented.
- A new, fourth HMM was designed and implemented. This model takes a simpler approach to chord to note modelling, repeating chord labels to create a one-to-one mapping between chords and notes, with a Multinomial emission model. See Section 3.1 for details.
- An LSTM model was designed and implemented. Initially intended to be a full sequence to sequence encoder and decoder model for varying input output lengths, it was instead decided (based on the advice of Adam Lopez) to implement a single one-to-one mapping LSTM model. See section 3.2.
- The development environment (primarily Jupyter Notebooks [57]) was then moved to a format that could run on the school's `msccluster` of compute nodes.
- The models were then properly evaluated with 10-fold cross validation.

Chapter 2

Theoretical Background

This chapter goes through the main concepts necessary to understand the design of the systems built in this project. Each section is intended to provide a brief summary of the necessary information on each topic to allow the reader to fully understand the workings of the models in the next chapter, as well as an understanding of the design choices.

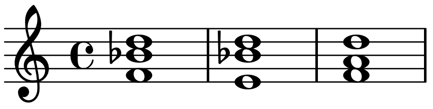
2.1 Music Theory

It is safely assumed that the reader is familiar with the concept of music. If not then it is suggested that they study the following albums before returning to read this report: Blood On The Tracks by Bob Dylan [16] and Songs From a Room by Leonard Cohen [10].

Music is fundamentally concerned with two concepts: rhythm and pitch. Rhythm is, at its simplest, a regularly repeating pattern. The periods of repetition that musical rhythm is most concerned with lie around the same frequency as that of the human gait, or heart beat (the reader is left to draw their own conclusions about the significance of this). Pitch is, essentially, also concerned with regularly repeating patterns, only now the domain is hundreds of beats per second. The human ear experiences this as a musical note, rising and falling in pitch as the frequencies increase and decrease.

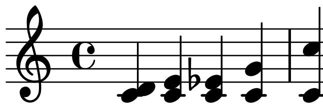
Music has been studied and practised by many different cultures over the course of human history and there have been many different formalisms developed for describing it. The most common, and that used in this report is *Western Tonal Music* (WTM). Under this paradigm, the continuous range of pitch is divided into repeating *octaves*, where corresponding points on each *octave* are perceived as the same note at different pitches. These ranges are then subdivided into 12 distinct notes. The method used for these divisions is based on ratios between note frequencies, the most commonly used being Equal Temperament [58]. Pieces of music are organised into one of 12 *keys*, each with a *tonic* corresponding to one of the 12 notes. Keys can be thought of as a prior over the distribution of notes in a song, where the tonic note has the highest probability of being played.

In written music, pitch is associated with vertical direction on a *stave*, shown below.

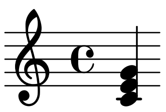


Rhythm is concerned with the temporal positioning of these notes, both where and when they occur, as well as for how long to play them. In WTM a piece of music has an associated beat or *tempo* that defines the frequency of an underlying, unheard pulse, over which the notes of the piece are laid, typically close to 120 beats per minute. The *time signature* of a piece defines how to treat groups of these beats, with the most common being $\frac{4}{4}$. This corresponds to groupings of four beats (also called *quarter notes* or *crotchets*). These groupings are referred to as *bars* and are used to organise the temporal structure of a melody in a way that is both easy to read and represents the underlying rhythmic properties of the piece. As in western languages, time flows from left to right in written music.

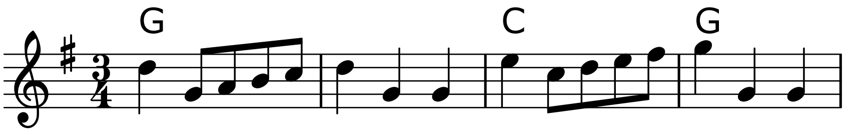
The above is a very brief overview of melodic structure in a piece of music. However there is another concept that is fundamental to this problem space, and that is *harmony*. Fundamentally, *harmony* is the relationship between the pitches of two or more notes, as they are heard together. It is concerned with the *intervals* between the notes and how they are perceived. The most basic interval is the *semitone*, the distance between one note and its neighbour, hence there are 12 semitones in an octave. The harmonic perception of an interval of two or less semitones is dissonant and jarring. The octave itself is an interval, which is perceived as resonant and uncharacterised, another way of saying that two pitches an octave apart are perceived as the same note. The next simplest interval is when two notes are 5 semitones apart, the *perfect fifth*, referred to as a *power chord* in guitar terminology, it adds the most basic harmonic colour. The intervals of 3 and 4 semitones are where harmony takes on an emotional quality, 3 semitones is referred to as a *minor third*, and evokes a feeling of darkness and melancholy, when perceived; whereas 4 semitones, the *major third* evokes the opposite emotion, that of brightness. It is this incredible connection between emotions and intervals that makes musical harmony so fascinating.



A *chord* is when more than one note is played at one point in time, typically at least three. The basic three-note chord is called a *triad*, composed of notes that are based on intervals, relative to the note in the first position, the *root*. The other two are the *perfect fifth* and a *third*, either *major* or *minor*. Depending on the *third* that is chosen, the full chord will take on one of these qualities. The following diagram shows the basic C major chord.



A modern musical arrangement, for example a jazz standard, typically consists of both a melody and an associated progression of chords, intended to be played as an accompaniment to the melody. The problem space of this project is concerned with the association between these two, specifically the problem of generating one if the other is not present. Generating a melody is what is typically considered musical composition, a difficult problem to solve on which much work has been done in recent years [11] [47] [31] [81]. This project concerns itself with the other direction, predicting chord sequences from melodies. This problem is conceptually half sequential classification, half generation, as there is not necessarily a unique chord sequence for every melody, but there is still a ground truth that can, theoretically, be derived from the observed notes. The following piece of music is an example from one of the corpora used for training the models.



In terms of a traditional classification problem, $f(x) = y$, the x here is a sequence of notes and the y is a sequence of chord labels. Hence we have a sequence to sequence problem, a class of problems that has seen a lot of attention in recent research [72] [41] [15].

2.2 Inspiration from Natural Language Processing

An often-drawn comparison is that of the similarity between music and language. Many postulate that music is itself a form of language [9] (although perhaps it is more enlightening to say that language is a music). This is useful when analysing music as there is a wealth of literature on natural language processing.

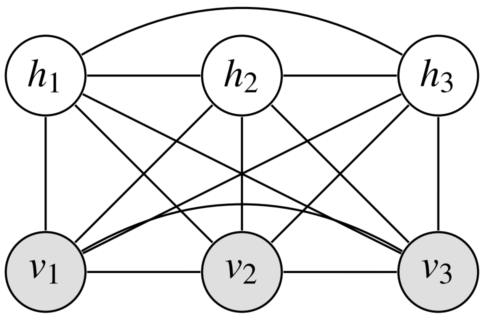
The immediate comparison that can be drawn is to Part of Speech (POS) Tagging. In this problem we are trying to assign grammatical tags to a tokenized sequence of words, for example:

And/CC now/RB for/IN something/NN completely/RB different/JJ

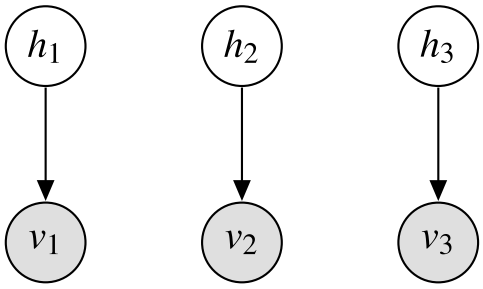
This one-to-one mapping problem is analogous to chord labelling in the case where each note is labelled with a corresponding chord:

c/Cmaj d/Cmaj e/Emin f/Emin g/Gmaj

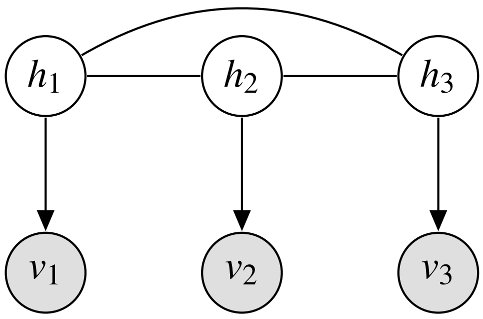
We can describe this in terms of latent variables h_i and visible variables v_i:



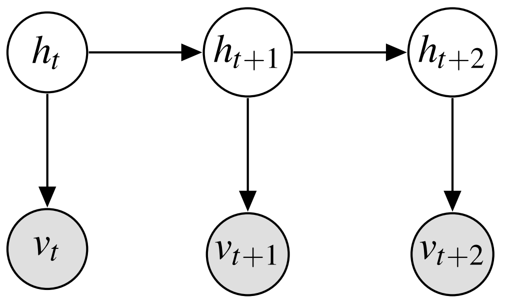
Both are examples of a problem where the visible variables that are observed are dependant on the latent variables that are trying to be determined. As can be seen in the diagram above, the naive approach introduces a large number of connections to model. This problem would be simpler if said latent and observed variables were not dependant on each other, and each observed variable only depended on one hidden variable:



But this is clearly not the case. An adjective is very likely to be followed by a noun, and a G major is very likely to be followed by a C major. In reality the situation looks more like this:



It is useful to assume independence of observed variables given the latent variables and this is something that all the models do, but the relationship between the h_i are more complex and this is one area where the difficulty arises in these problems. An often used approach to this is to apply the first-order Markov Assumption. This allows us to assume that each latent variable h_t is only dependant on the previous latent variable h_{t-1} in a temporal ordering $t \in \{1 : T\}$. This model is known as a Hidden Markov Model (HMM). POS Tagging is a success story for the HMM [36].



2.3 Hidden Markov Models

The HMM assumes an underlying latent state space $\mathcal{H}$, with transition probabilities between members. These variables are conditionally independent of each other, given the previous h_{t-1}, i.e. the first-order Markov Assumption:

$$P(h_t|h_{t-1}), \quad h_t, h_{t-1} \in \mathcal{H}, \quad t \in \{1:T\} \tag{2.1}$$
$$I(h_t, h_i|h_{t-1}), \quad i \in \{1:T\} \tag{2.2}$$

And a visible variable space $\mathcal{V}$, which are conditionally independent of each other given the corresponding latent variable at time t:

$$P(v_t|h_t), \quad h_t \in \mathcal{H}, \quad v_t \in \mathcal{V}, \quad t \in \{1:T\} \tag{2.3}$$
$$I(v_t, v_i|h_t), \quad \forall i \in \{1:T\} \tag{2.4}$$

Given these assumptions we can write the entire joint probability distribution as:

$$P(\{v_t\}_{t=1}^T, \{h_t\}_{t=1}^T) = P(v_1|h_1)P(h_1)\prod_{t=2}^{T} P(v_t|h_t)P(h_t|h_{t-1}) \tag{2.5}$$

Which simplifies the complexity of the problem enormously.

There are various forms of inference that can be performed on HMMs. In the case of on-line accompaniment prediction, the probability we are interested in would be $P(h_{t+1}|\{v_i\}_{i=1}^t)$. This can be performed by first summing over all possible states at time t, then calculating probabilities of those h_t by propagating recursively back to the start state. This is referred to as the forward algorithm or *filtering* [61]. In terms of message passing in graphical inference, the message is being passed from all observed variables $\{v_i\}_{i=1}^t$, starting at $t = 1$, summing over the latent variables h_t as the message is passed up the chain.

The case looked at for this report is concerned with the off-line variant of this problem, predicting the most probable sequence of latent variables, given the full observed sequence:

$$\arg\max_{\{h\}_{t=1}^{T}} P(\{h\}_{t=1}^{T}|\{v\}_{t=1}^{T}) \tag{2.6}$$

This is calculated by a form of dynamic programming called the Viterbi Algorithm [61].

So this model allows us to generate the most likely sequence of hidden, latent variables given an observed sequence, with the assumptions that each observed variable is only dependant on the others *through* its corresponding latent variable, and that each latent variable is only dependant on its past *through* the previous time step, and is entirely independent of its future. See section 3.1.1 for a discussion on the problems these assumptions raise for this problem space.

2.4 Neural Networks

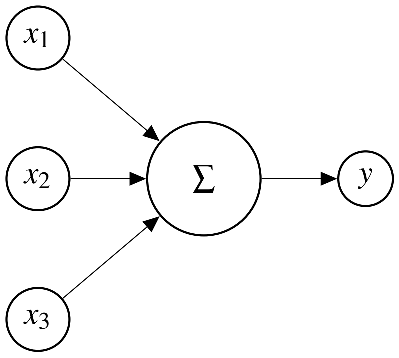
One immediate concern with the HMM is that it is unable to capture long term dependencies. It is memoryless, or, more accurately, can only remember one thing: the last place it visited. This contradicts our intuitive grammatical understanding of language. A grammar is a tree structured language model, where the following section can be very dependant on something that happened at the start of the sequence. One example of this in music is repeated phrases; you can't repeat something if you have forgotten it.

What we want is a model that can carry information through it then learn when to forget it and when to incorporate it into deciding the output at a given time. A popular model for this type of system is the Recurrent Neural Network (RNN), in particular with a Long Short Term Memory (LSTM) cell.

Work on neural network architectures has flourished in the past few years, due largely in part to advances in optimised computation on GPUs [49][33], as well as new pre-training methods such as autoencoders [27][76][13]. However the basic concept has not changed very significantly since its beginnings in Hebbian Learning and the Perceptron in the 1940s and 1950s [26] [60].

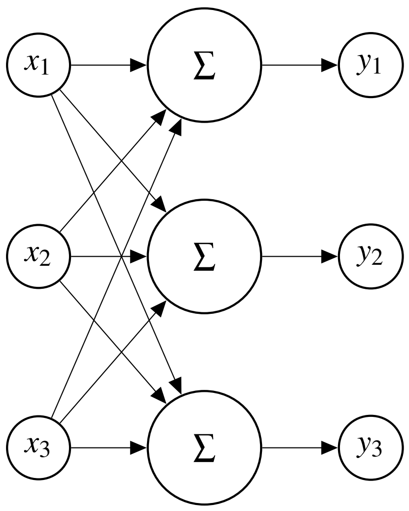
2.4.1 Perceptrons and MLPs

A perceptron is a linear classifier based on an abstraction of a neuron. It is, in essence, a weighted sum of inputs with one output.



The key observation is that these weights can be trained to create a binary classification model, where the two classes are linearly separable. The classic example of a problem this model cannot solve is the XOR logic gate, as in this case a straight line cannot be drawn which separates the two classes [44].

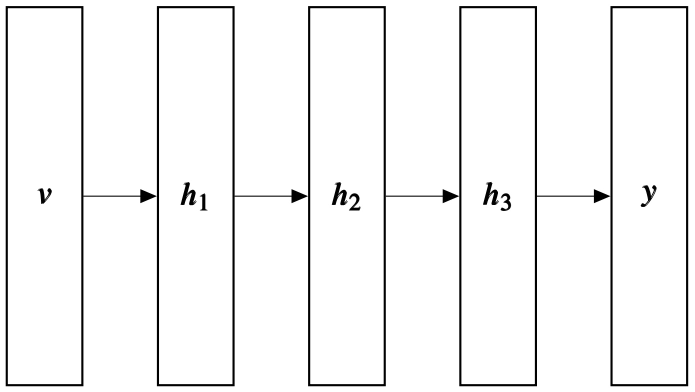
If we take a row of perceptrons with the same inputs but different weights, we now have one layer of a Multi Layer Perceptron (MLP). MLPs are built by taking the outputs of these layers, applying an activation function such as a logistic sigmoid (σ) or a hyperbolic tangent ($tanh$), then treating these values as the inputs to another layer.



Another name for these models are Feed-Forward Artificial Neural Networks (ANNs). By continuing like this we can build an arbitrarily large ANN, which can be used to model more and more complex problems. In fact it can be proven that even just a single layer, finite length ANN can be used to approximate any continuous function on compact subsets of $\mathbb{R}^n$ [12][30]. This astounding property, known as the Universal Approximation Theorem, is one of the reasons that neural networks are outperforming many traditional mathematical models currently, in a variety of fields from computer vision [33] to protein modelling [75].

However this theorem gives no algorithm for building these complex models. The difficulty comes in training the weights to correctly model the problem, and in the case of very deep models this can mean millions of parameters. Even worse, there is no analytic solution in optimising these parameters, they must be trained iteratively. This means a large amount of computation time and resources.

To train an ANN in a supervised manner, an error function is needed between the output of the network and the training labels. To continue the notation introduced in the HMM section we will let the $\{v_i\}_{i=1}^{T}$ be the input to the network, $\boldsymbol{v}$ using vector notation. The output variables are re-named $\{y_i\}_{i=1}^{S}$, or $\boldsymbol{y}$, to avoid confusion with the hidden layers of the network, $\boldsymbol{h}_i$, for the ith layer. Let the training label associated with $\boldsymbol{y}$ be denoted $\boldsymbol{t}$. A superscript in parentheses, $\boldsymbol{y}^{(j)}$, will be used to denote the jth instance of data, from a data sample of size N. In the case of three layers, this will look like:



Hence we have an error function:

$$\sum_{j=1}^{N} E(\boldsymbol{y}^{(j)}, \boldsymbol{t}^{(j)}) \tag{2.7}$$

The weights are then updated in a way that minimises this error function. To do this it is necessary to take the derivative of the error with respect to each weight. This is simple to do with a single layer but becomes more complex with a deep network. The method for achieving this is called Back Propagation and was created by Paul Werbos in 1974 [79]. Methods for iteratively updating the gradients based on these derivatives are called Gradient Descent and tend to take the following form:

$$w_{ik} := w_{ik} - \eta \frac{\partial E(\boldsymbol{y}, \boldsymbol{t})}{\partial w_{ik}} \tag{2.8}$$

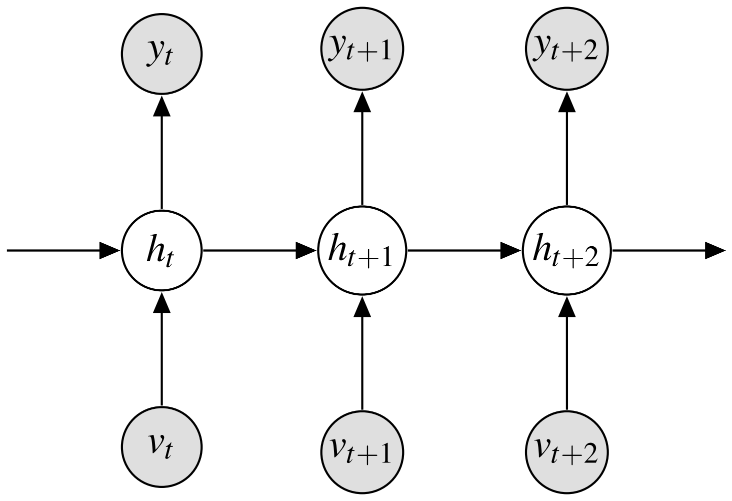
Where w_{ik} is the kth weight of the ith layer. These gradients can either be done per data point j or by summing over batches as in 2.7. By this method the neural network can be slowly pivoted towards on optimal configuration, but there is no guarantee of avoiding local minima, and even if it manages to perfectly fit the training data, it will likely be overfit and not generalise to held out test data. Despite these problems, advances in processing speeds and hardware level algorithms [34] have made training these models possible.

2.4.2 Recurrent Neural Networks and LSTMs

So is it possible to use neural networks to predict a sequence of chords from a sequence of notes? Unfortunately, in the current state described, the ANN takes a vector $\boldsymbol{v}$ of fixed-size T as input, and outputs $\boldsymbol{y}$ of fixed size S. What is needed is a network with a flexible input and output size, a tricky problem under the current architecture as this would mean adding and removing weights for every data sample. Recurrent Neural Networks (RNNs) solve exactly this problem.

An RNN is a neural network that scales dynamically to its input. It is composed of *cells* that share weights and are connected together in a chain. A single RNN cell looks like a standard feed-forward network, but its input v_t is combined with an output from the previous cell h_{t-1}. Architectures vary but it is common for the output of the cell, y_t, to be the output of a second layer that takes h_t as input, and for the output of this hidden layer, h_t, to be passed to the next cell in the chain. Of course these cells can be further stacked to result in deeper architectures that learn higher-level features of their input.

The following diagram represents a simple, one-layer RNN, where the full layers are now represented as circular nodes, for compactness and comparison to the HMMs (note in the HMMs the arrows represent dependence relationships, here they represent weighted sums and activations, but the concept is similar):



To train these models we can unfold them after receiving all of the input, so that it looks like a complex feed-forward model, that back propagates the error in a similar method to before, now called Back Propagation Through Time (BPTT) [80].

A problem with this method, referred to as the vanishing gradient problem [28], is where the earlier weights in the network receive a much smaller update than those closer to the final output, as the back propagated gradient at time t has been through $T - t$ applications of the chain rule, each involving multiplication with numbers < 1, resulting in the gradient becoming incredibly small. Hence errors caused by weights near the start of the sequence won't be corrected.

One proposed solution to this is the Long Short Term Memory (LSTM) Cell [29]. This is a more complex cell structure involving 3 *gates* and an explicit cell state, c_t, that serves as the system's memory. Each gate combines the input and the cell state in a way that serves an intuitive purpose.

The first gate is called the *forget gate*. It takes as input a concatenation of the input v_t and the output of the previous hidden layer h_t, performs a weighted sum, and uses a logistic sigmoid to output a value between 0 and 1 for each component of the cell state c_t. Through element-wise multiplication, this is used to decide which elements of the cell state to forget at this time step. This is the equation for the input gate before updating the cell state (note that bias terms are omitted).

$$f_t = \sigma(W_f[h_{t-1} : v_t]) \tag{2.9}$$

The next gate is the *input gate*. This is used to determine what new information to add to the cell state. The components to update are chosen in a similar manner to above, then the *tanh* function is used to generate new updates from the input.

$$i_t = \sigma(W_i[h_{t-1} : v_t]) \tag{2.10}$$
$$c_t^* = \tanh(W_c[h_{t-1} : v_t]) \tag{2.11}$$

These updates are then applied together to the cell state.

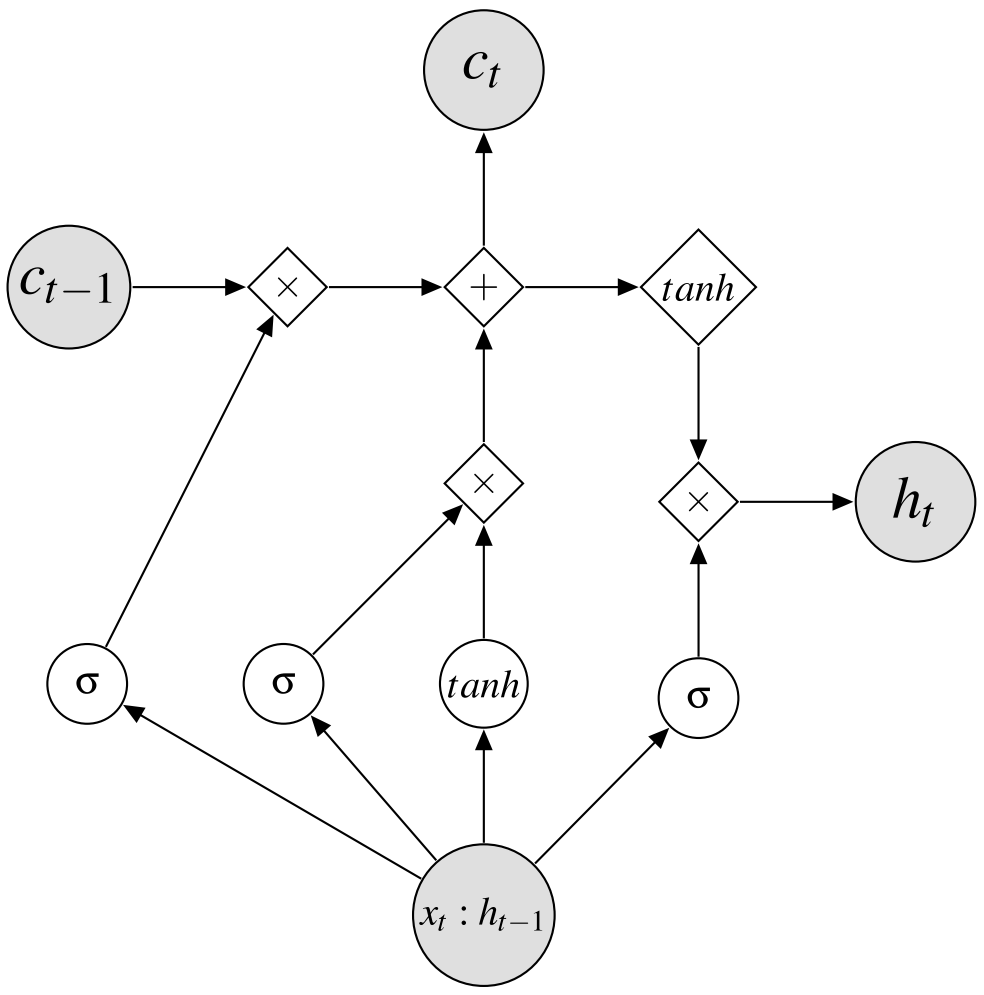
$$c_t = f_t * c_{t-1} + i_t * c^* t \tag{2.12}$$

Where here $*$ signifies element-wise multiplication.

Lastly the output gate decides what to output as h_t for this time step. It chooses components by the same method as the previous gates, then generates the output from the cell state.

$$o_t = \sigma(W_o[h_{t-1} : v_t]) \tag{2.13}$$
$$h_t = o_t * \tanh(c_t) \tag{2.14}$$

The full cell takes this shape:



This cell represents a single layer LSTM model, where the white circles are neural network layers with an activation function, the diamonds are pointwise operations and the dark circles input/output states. This could be used as the full model, in which case the h_t would be the outputs y_t of the system. Alternatively these LSTM cells could be stacked, creating a deep learning model where the h_{it} of each layer i would be the input to the layer $i+1$, where h_{1t} are the original inputs v_t.

All this serves to create a model where information can be remembered and forgotten dynamically based on where we are in the input sequence. This is very useful for the purposes of this project as repeated musical phrases and structures could be implicitly captured by the model, as shown to be possible by [18].

LSTMs are currently the cutting edge for many different areas of natural language processing, including language modelling [71] [55], speech recognition [25] and language generation in spoken dialogue systems [78].

For an in-depth discussion on LSTMs, [50] or [24] is recommended.

2.5 Review of Previous Work on Automatic Harmonic Analysis

A brief review of previous work on chord progression estimation and related works will be presented in this section. Note that this is largely unchanged from the report presented last year.

The field of automatic harmonic analysis of music has seen a wide range of research in the past few decades. Much of this stems from the interpretation of music as a form of language, or at least recognising the similarities between music and natural language and exploiting them to achieve certain tasks. The works of [83] and [21] were some of the first applications of computational linguistic theory to the field of music, inspired by the pioneering work of Noam Chomsky on the formal definitions of languages and syntax [40]. In [83] a generative grammar for the parsing of a harmonic phrase was proposed, using an adaptation of the formalism of tonal harmony proposed by [20]. This mechanism was capable of automatically parsing chorales.

An alternative method of harmonic analysis, introduced by [63], was used in another computationally assisted, automatic parsing of music in [68]. Similar to the above, Smolier applied existing NLP parsing techniques to a grammatical formalism of tonal music.

Logic based approaches have also been attempted. In [17] an expert system for the harmonic analysis of chorales is proposed, based on first order predicate logic. This was expanded on by [42], where a full approach applicable to all tonal music was presented. A similar hierarchical logical representation for the computational analysis of music was proposed by [67].

The development of probabilistic machine learning algorithms allowed for a more statistical approach to harmonic analysis to be developed. The work of [37] applies neural networks to chord classification, focusing on the problem of how to best represent pitch. A more qualitative perception model for musical learning is presented in [82].

A popular and successful method of probabilistic harmonic analysis is the HMM based approach. The harmonic analysis of chorales proposed in [3] uses HMM probabilistic inference. In [39] an HMM based method for chord generation from a hummed input is presented. A simple frequency count based HMM is used in the chordal analysis of the MySong application by Microsoft [66,], a system that automatically generates an accompaniment for a real-time vocal input. A more complex HMM based system is utilised by [62] in the automatic transcription of melody and chords from the first eight Beatles albums.

In more recent years, a template based approach to chordal analysis has been proposed, notably in the work of [53], [52] and [51]. This method moves away from more statistical techniques and instead assigns a rule-based scoring to a frame, based on whether the observed notes are present in the stored chord model template.

More complex hybrid approaches have also been attempted. A combination of neo-Riemannian transformations, Markov chains and Support Vector Machines are used in [8] for the generation of style-specific accompaniment. This method uses machine learning techniques to decide how probable it is that an observed note is a ’chord tone’ for the current chord and is the inspiration for one of the proposed emission models used in the previous year of this project.

It should be noted that the majority of these approaches assume pre-segmentation of the input into distinct chord segments for classification, as does the work presented in this report. A complete model of chordal analysis must take this aspect into account as well. There are many well researched methods for accomplishing this, a good review of a few of these is provided in [53].

Chapter 3

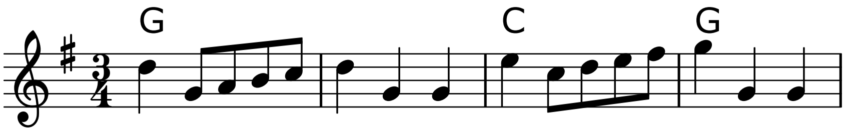
Design and Implementation

The aim of this chapter is to take the reader through the design of the models, by way of example on a piece from the KP Corpus, and to explain the details of the implementation. The models based on HMMs differ in structure from the LSTM models so there is a section for each of these.

3.1 Hidden Markov Model Based System

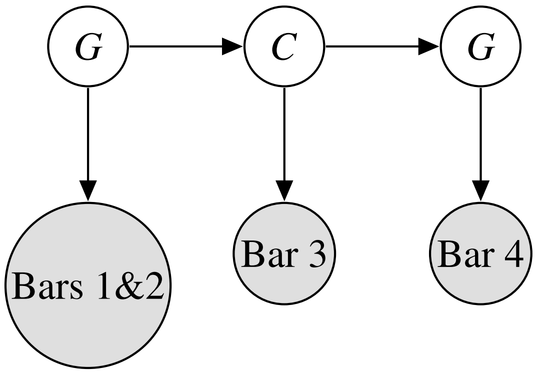
3.1.1 Design

The first year of this project was focussed on building an HMM based approach to the problem of chord labelling, with a view to use it as comparison to a model that was able to capture a higher level of structure this year. This section will briefly discuss the design of the HMM system, with a focus on a new design that was implemented this year, see [43] for an in depth description of the previous year's implementations.



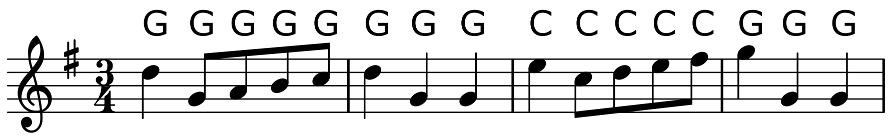
Above is the example piece from the previous chapter, Minuet in G (often attributed to Bach). To reiterate, the task is to take as input the sequence of notes in both staves and to predict the sequence of chord labels. The first thing to notice is that there is not a one-to-one mapping between notes and chord label, there are many more of the former. If we let the chord labels be the latent variables $\{h_t\}_{t=1}^T$ of our HMM and the notes be $\{v_t\}_{t=1}^T$ (as per the notation used in the previous chapter, where T is the number of time steps and t is the time step of the current note or chord), then v_t represents a group of notes that is generated by one chord. This concurs with our intuition of the problem, where in essence we are using the HMM to model the 'generation' of notes by a sequence of chords (whereas in practice we will run this model in the opposite direction to obtain the most probable chord sequence). We can think of this as a song generating machine that can be in a number of states, each state representing a chord.

This starts in some state and outputs of group of notes, then transitions to another state and outputs a new sequence of notes. In our example the first state would be the chord of *G*, and the first group would be the first two bars of notes, where the second state is the chord *C* and the second group is just the third bar of notes.



The models of the first year of this project used exactly this assumption and addressed the problem of going from a sequence of note groups to a sequence of chords. The inherent difficulties in this are that, for each state in the HMM, a smaller model must be constructed to capture the generation of this note group. Three approaches were used, one being a smaller HMM for each step, one being a 'bag of notes' model, analogous to a 'bag of words' model in natural language processing [84], and another more complex decision tree model. However these models do not capture how to split the observed notes into groups to start off with, so some pre-processing was assumed and necessary in the experiments. This is unsatisfactory as it does not provide a full model of the problem. Another, more fundamental problem is that the above assumptions discard a crucial indicator of the chord transitions. That is, the transitions between the notes at the end of one group and the beginning of the other. In the example, on the transition from the second chord *C* to the third chord, another *G*, there is a transition from an *F#* note to a *G* note one semitone above it. This is a key indicator of a chord transition as the melody is stepping up to the root note of the chord from the note below. This is something that would be beneficial to capture in the model.

Hence a new design of the HMM system is proposed and implemented this year, before the step to LSTMs. Consider a modification to the above machine. This machine outputs a sequence of notes one at a time, starting in some state. The time steps now correspond to each note, not to each chord, hence the machine is essentially self-transitioning for each note in the chord group. Then at some point the machine will transition to a new state and start generating more notes, but from this new state. The task is now to take the full sequence of notes and to try to determine to most likely sequence of states this machine was in when it generated them. Note that we now have a one-to-one mapping between the sequence of notes and the sequence of chords, see the re-labelling of the example below:

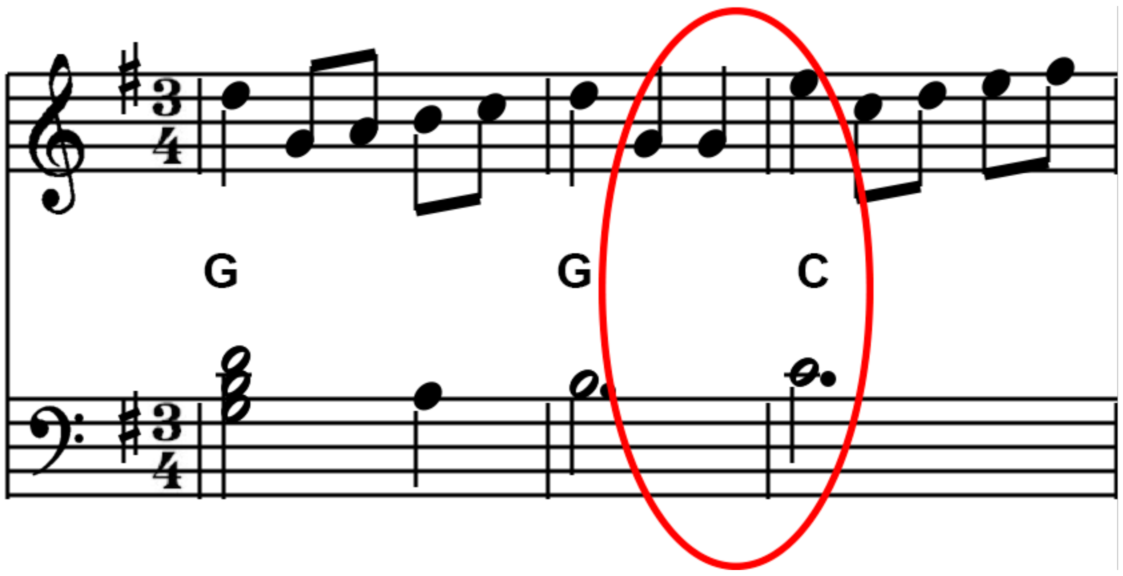


Now the v_t each correspond to a single note, and the probabilistic model of $P(v_t|h_t)$ is a simple multinomial distribution that can be observed from the training data, instead of a complex mini-sequence generation task.

It is important to note that we now have to address the problem of polyphony, where two notes are played at the same time. Our HMM model requires an ordered temporal sequence of notes, hence this effective chord in the melody must be broken down. This can be achieved by simply converting the chord into an *arpeggio*, which is a chord played one note at a time. For this model the order of the arpeggio is broken arbitrarily (in the MIDI implementation the order of notes as observed in the MIDI file is preserved).

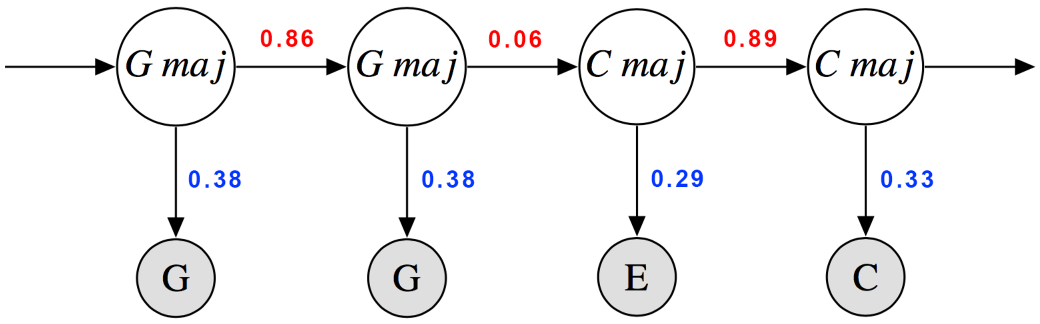
For illustrative purposes, the probabilities learned from an HMM model will now be analysed on the example. Note that this HMM model has already 'seen' this excerpt before in training, we will be analysing this overtrained model here just to illustrate what the model is able to capture. The data trained on is also untransposed, so the probabilities represent absolute relationships.

The section of the excerpt that will be analysed is where the model correctly determines the transition from $G\,major$ to $C\,Major$. Note that in the above description of the song the bass clef was omitted for simplicity, now shown below:



The notes in the red circle are represented in the following section of the song's HMM.

The numbers in red represent the probabilities of transitioning between each state and the numbers in blue represent the probabilities of emitted each note, given the chord.

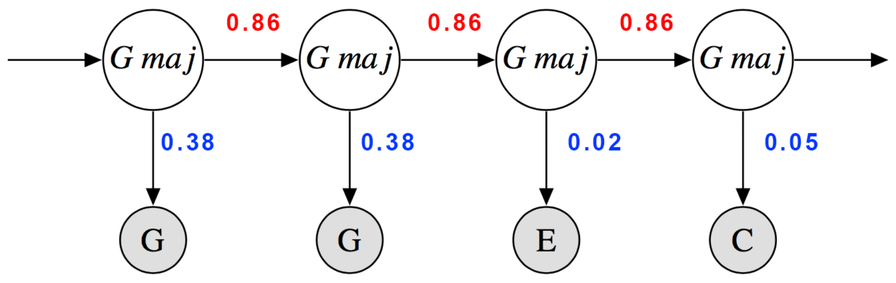


From this we can see that the model has learned that self transitions between chords are very likely, a result of the repeated labelling of notes with chords. This trade off can mean that the model finds it difficult to transition between chords and will prefer to try and make the sequence fit to one chord. However, in this case, the model effectively transitions to the chord $C\ major$, as this chord is much more likely to emit the notes E and C, and the transition to $C\ major$ from $G\ major$, although small, is the next most probable transition. The transition probabilities from a chord with root G are shown in table 3.1.

Table 3.1: Transition Probabilities from G

Root of Next Chord	Transistion Probability
G	8.6×10^{-1}
C	6.0×10^{-2}
F#	3.6×10^{-2}
D	2.5×10^{-2}
E♭	2.4×10^{-2}
A	1.5×10^{-2}
E	1.2×10^{-2}
F	1.0×10^{-2}
B	6.3×10^{-3}
A♭	3.1×10^{-3}
B♭	7.3×10^{-5}
D♭	3.3×10^{-5}

We can see, from the following alternative chord labelling, the probabilities of emitting the notes E and C under staying with a $G\ major$.



One of the difficulties this model faces is when the probabilities of emitting the notes under the correct chord do not out-weigh the bias of staying with the same chord.

It is important to note that the most probable sequence is determined for the song as a whole using the Viterbi algorithm, so the model takes into account products of these transition and emission probabilities and does not take the local maximum at a given time step.

3.1.2 Implementation

To implement the Hidden Markov Model used in the previous year's work, it was necessary to be modular in approach, with a re-usable high level structure of state transitions (the Transition Model) and a pluggable variety of chord state to note group models (the Emission Model). This allowed for the implementation and testing of the three main emission models described above. However, the majority of HMM libraries that exist in Python, the chosen language for this project, are restricted to basic Gaussian and Multinomial emission models [38] [5]. For this reason the decision was made to implement a more general HMM class that could be modified for each case of emission model. This involved implementing the Viterbi Algorithm [19], a dynamic programming method of determining the most likely sequence of hidden states in an HMM.

Note that this implementation of HMMs was purely for supervised learning, where the hidden states are known in the training data. This is fine for the high level structure where HMMs are used to predict chords, but for one of the emission models HMMs were used for the note groups at a lower level as well, where the hidden states are not as clear. In the previous year a second level of labels was added for this purpose, based on a system that measured how likely the note was to appear in a version of the chord (c.f. chord tones). For this year of the project one new implementation that was added was an unsupervised HMM that could approximate these hidden states on its own, using the expectation maximisation (EM) algorithm [45]. As this was only required for lower level use, with a multinomial emission, the `hmmlearn` [38] Python library was used.

The major alteration to the HMM model, discussed in the previous section, also had a relatively simpler emission model, compared to the previous year, and so the `seqlearn` Python library [5] was also made use of, for the supervised training of the model, allowing the system to benefit from optimisation that was not possible in the previous implementation of supervised HMMs.

For this new supervised HMM it was also necessary to replicate the observed chord labels for each note in the data. One small design decision that was necessary was the case of polyphonic melodies, where more than one note is played at once. For this case it was sufficient to treat these notes as being played after each other in quick succession, with order being split arbitrarily. In this sense the chords in the melodies are being split into arpeggios.

One key problem in the implementation of these systems is how to model sequential data. One method for dealing with this problem is to concatenate the sequences into a single dimension and to store this as a Numpy array [48]. For this implementation each note is stored as size 12 one-hot vector, where each index corresponds to a note pitch and the vector contains zeros at every index except that corresponding to the note it encodes. In this way we are treating notes as categorical data, avoiding the inherent ordering of storing them as integers. Hence we can store all the data as a 2 dimensional, $D \times T$, Numpy array, where D is the number of notes (12) and:

$$T = \sum_{i=1}^{N} t_i \tag{3.1}$$

where N is the number of songs and t_i is the number of notes in song i. So the sequences of notes in each song have been concatenated together into a single data structure. A similar operation is done for the chord labels, of which there are also 12 as, for the purposes of experiments, only the roots of the chords are chosen as the labels, see section 3.3.

Another benefit of storing data as Numpy arrays is the benefit of optimised matrix operations, in contrast to index iteration for the implementation of certain algorithms. This doesn't affect the HMM model but is very important when using neural networks see section 3.2.2

3.2 Recurrent Neural Network Based System

We now move on to the LSTM based system and the design decisions that were involved in building it.

3.2.1 Design

The HMM is a very good approximate model with an intuitive understanding but it's main drawback is the first order Markov assumption. We know from music theory that there is an underlying structure in the design of chord progressions [35] [56]. One possible approach for making use of this structure would be to incorporate a grammatical 'language' model of chord progressions, as constructed by [23] or [59]. The approach taken in this project was to instead build a system that could learn an implicit representation of structure with memory. This was inspired by the success of such systems at learning language models in other applications, see [64] [71]. Hence the decision was made to build an LSTM based sequence to sequence model.

In general a sequence to sequence model refers to the combination of an encoder and a decoder, taking an input sequence to an output sequence, each of which can have a different length. This would have been possible to implement for this problem, but due to the direct local relationship of chords to notes it was decided that a simpler system would be more beneficial. This design makes use of the repeated representation of the data, where there is a one-to-one mapping between chord labels and notes. We can then use a model where each LSTM cell's output is treated as a chord in the prediction sequence.

Instead of only passing the information of which chord was last played to the next state, this model allows for the passing of abstract learned information from time $t-1$ to time t. This allows the system to model a more complex, context-dependent relationship between the current note and the current chord. For example the note E on its own is not very likely to be played over a $D\ major$ (the triad notes of a $D\ major$ are D, $F\#$ and A, an E would be dissonant), but if it's preceded by a D then it could be an intermediate note between D and $F\#$. This relationship can be implicitly captured by the Viterbi algorithm for HMMs in the prediction of the most likely sequence (the chord sequence is not likely to jump to an $E\ major$ for a single note then back again) but in this way this information can be directly encoded in the LSTM model. Something that an HMM might not be able to capture is that if, say, the chord progression $G\ major$, $G\#$ $diminished$, $A\ minor$ occurred at one point in the song, then if $G\ major$, $G\#\ diminished$ at some point later then a very probable next chord is the same $A\ minor$. Concepts like repeated structure (at note level too) and long term dependencies are theoretically able to be captured by the LSTM. In practice it is very difficult to determine whether this has happened, as the information is encoded in the weight matrices of the LSTM cell, not an easily readable probability table.

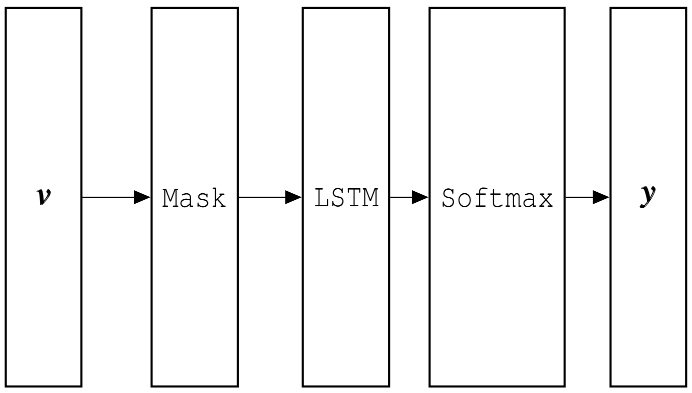
3.2.2 Implementation

In recent years, new open source tools have been made available to facilitate the implementation of machine learning (ML) algorithms. One notable example is Tensorflow [1], a highly optimised platform for creating ML architectures as a computation graph in C++, with a Python interface. Tensorflow has a symbolic, declarative structure. Variables and interactions between them such as matrix multiplications (a key operation in neural networks) are defined fully before any computation on data occurs, then a session is opened and the data is passed through the computation graph. This has proven to be a very powerful tool and it is now widely used in the field.

A higher level, specifically neural network aimed Python API exists, built on a Tensorflow backend, called Keras [7]. This framework abstracts away low level implementation of weight matrices and gradient calculation to allow design at the level of neural network layers and declarative algorithm choices. This library has proved very useful for research as it allows for quick, reliable design and experimentation on neural networks. These two libraries, Keras with a Tensorflow backend, were used in the implementation of the LSTM model.

After general preprocessing (see the next section), it was necessary to transform the data into a form that could be processed by the LSTM model. This is the same problem of modelling sequential data faced by the HMM. An alternate solution was used on this model, one in-line with the guidelines of the libraries used. Instead of concatenating the data into one dimension, it is more beneficial for a system based on matrix multiplication, such as neural networks, to have more smaller dimensions, over-which these operations can efficiently be performed. Accordingly, our previously 2-dimensional data (pitch $\times$ full note sequence) is now modelled as a 3-dimensional tensor (pitch $\times$ song note sequence $\times$ songs). However this now poses a problem: The pitch dimension is always size 12 but there are a varying number of notes in each song. The solution is to simply pad the ends of each song with a special *PAD* vector so that every song is the length of the longest song, which will have no padding, where the *PAD* vector is implemented as the zero vector. This raises problems of its own, such as efficiency of checking the maximum length, and transferability to an on-line setting (how can you know if a longer song might appear next?), but it is suitable to the purposes of this project.

This padded data is then fed into the first layer of the model, which is a masking layer that looks for the padding and masks it from influencing the rest of the layers while maintaining dimension size. This is, helpfully, a provided feature of Keras. The next step is the main LSTM layer, implemented with dropout to avoid overfitting on the training data, the hyper-parameters of which experiments were run over. Each LSTM cell in this layer outputs a vector which is then fed into a time distributed fully connected layer with softmax activation. Essentially this is a standard, dense neural network layer that takes the outputted vector of the LSTM cell to a K dimensional vector of probabilities over each chord, where K is the number of chords (typically 12 in this study). This dense layer is then repeated for every time step in the sequence, as the LSTM cell is a layer below. These softmax output logits are then used to predict the most likely chord, akin to the multinomial output of the HMM.



The model described above is the most simple of those used, for experiments on more complex structures see chapter 4.

The next step is to train the model. The loss function used for this training was the categorical cross-entropy between the logits and the true labels [65]. This is a method for measuring the 'distance' between two probability distributions (the labels can be considered a probability distribution with certainty on the true value) and can be defined as follows:

$$E(\boldsymbol{y},\boldsymbol{t}) = -\sum_{i=1}^{T}\sum_{k=1}^{K} t_{i_k}\log(y_{i_k}) \tag{3.2}$$

The gradient descent algorithm used for training was Root Mean Square Propagation (RMSProp), an idea proposed by Hinton and Tieleman in 2012 [73]. The idea here is to, for each weight, divide the learning rate by a factor proportional to a running average of the size of that weight's gradient:

$$w_{i_j}^{(t+1)} = w_{i_j}^{(t)} - \frac{\eta}{v(w_{i_j},t)}\frac{\partial E(\boldsymbol{y},\boldsymbol{t})}{\partial w_{i_j}} \tag{3.3}$$

where $v(w_{i_j},t)$ is the running average of the gradient size (implemented with a forget parameter) and η is the learning rate. Note that in the model there are more than just two indices needed to identify all the weights, i and j here are used to simplify illustration.

Another small consideration that was needed was to carefully weight the cross entropy loss to avoid taking the padding into account. Astonishingly high faux-accuracies and over-early celebrations were almost avoided.

One advantage of the Tensorflow backend is that it is highly optimised to GPUs. The more complex models with more weights were trained on the university's compute `msccluster` which has access to 3 Geforce Titan X Nvidia GPUs in each node.

3.3 Preprocessing

This section briefly details the general pre-processing that was necessary for converting the data from its raw state to a form that could be processed by the models. The datasets used for the experiments are described in detail in section 4.1 but each provides data in very different forms, so each will be broken down here.

3.3.1 Weimar Jazz Database

The Weimar Jazz Database (see 4.1.1) provides a graphical user interface packaged with pre-processing tools for the extraction of features from its dataset. The data used for this project in the first year made use of many of these features (metrical information etc.) but the models of this year abstract the melody as a simple sequence of 12 possible pitch classes (i.e. C through B♭). Hence the data was exported as a `csv` file of pitch classes with matching chord labels as strings. Note that these pitch classes are transposed to the same key, hence a 0 in the key of *C* will be a *C* note, but a 0 in the key of *G* will be a *G* note. These pitch classes are hence called Tonal Pitch Classes (TPCs).

The transposition of all songs to the same key (the key of *C*) is beneficial to the design of all models as it allows for relative relationships to be learned, not just absolute relationships. Hence this is included in the design of the system, although it should be noted that in some cases the models make 'more interesting' mistakes without transposition, see the results section.

The chord strings are then converted pitch classes, and then transposed to TPCs, to be used as chord labels. All TPCs were then converted into one hot vectors. The reading of the `csv` files, preprocessing and the collation into arrays for further processing is packaged into a `DataLoader` class.

3.3.2 KP Corpus

The data from the KP Corpus (see 4.1.2) is provided in Musical Instrument Digital Interface (MIDI) format. MIDI files consist of `note_on` and `note_off` events of varying length and pitch, arranged into a list, plus some meta data. The chord labels for this corpus are provided as annotated lyrics in the MIDI files.

The `midicsv` [77] utility was used to convert the MIDI files to `csv` format, with no other processing, allowing the `note_on` and `note_off` events to be read into Python. More complex preprocessing was then required to generate a sequence of TPCs, implemented in a separate `DataLoader` class for this dataset. Transposition and chord label extraction was done in a similar way to the previous Weimar Jazz Database.

Chapter 4

Experimental Results

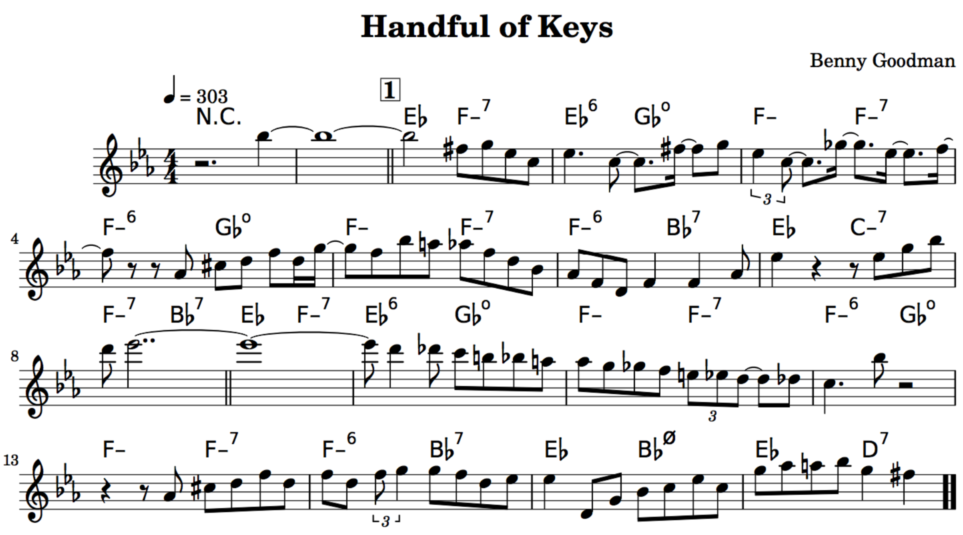
In this chapter the datasets used, experimental design and main results of the project will be presented and discussed.

4.1 Datasets

A key problem with analysing the harmonic structure of melodies is the availability of data. Primarily due to copyright issues, much of the data used by similar projects in the literature is not made available publicly. What is available is largely just melodic, without chord annotations [74] [46]. Some purely harmonic datasets also exist [6] [14], but this task requires the aligning of the two for training and analysis.

4.1.1 Weimar Jazz Database

The dataset used for the previous year's work was the Weimar Jazz Database [2]. This is a collection of 273 transcribed solos from famous Jazz musicians, typically over a Jazz Standard (a commonly known piece of jazz music over which a group of musicians will improvise). The meta data in this data set is extensive and of a very high quality, but inherent difficulties come in the domain of melodies. Improvised jazz solos are, in a music theoretic sense, very noisy. They tend to be unpredictable and ambiguous harmonically, increasing the difficulty of this problem. Below is an example of one of the pieces in the database.


Handful of Keys
Benny Goodman
= 303
N.C.
1
E♭ F-7 E♭6 G♭o F- F-7
F-6 G♭o F- F-7 F-6 B♭7 E♭ C-7
F-7 B♭7 E♭ F-7 E♭6 G♭o F- F-7 F-6 G♭o
F- F-7 F-6 B♭7 E♭ B♭ø E♭ D7

4.1.2 KP Corpus

The first main task of this year's work was to find a new database that contained simpler melodies, on which the models of last year could be retrained on and on which the new models could be developed. For the reasons stated above this was a difficult search. The dataset that was settled on was a collection of excerpts from classical songs in MIDI format, called the KP corpus as it was created by Kostka and Payne in accompaniment to their workbook on Tonal Harmony [32]. These melodies are much simpler and easier to analyse, with a stronger correlation between notes and chords. The example used earlier, Minuet in G (shown below) is from this dataset. The only downside of this corpus is that there are only 46 excerpts, a small number for training and evaluating stochastic models. The piece used earlier for analysis in this report, *Minuet in G*, is an example of one of the excerpts in this dataset.

4.2 Experimental Design and Accuracy Metric

It is important to note that evaluating these models is difficult, as the nature of the problem is inherently ambiguous. The chord in the data may say *C major* but perhaps an *A minor* would work equally well. Unfortunately there in no easy way of encoding this into an accuracy metric for our model. A substitution table could be imagined but this would be highly subjective and too lenient on an accuracy metric. For this reason a hard, note by note chord comparison was used to evaluate the accuracy of the models e.g. the accuracy of a produced chord sequence is the percentage of positions where it guessed exactly the ground truth chord.

Another key problem is the choice of what to use as chord labels. Previous in this report the chords have been referred to either by their root or their root plus a *major/minor* modifier. But in reality chords can take on a variety of characteristics including including *dominant* $7ths$, *suspended* $4ths$ and many others. Including all of these as individual categories would quickly reduce any accuracy measure to zero. For this reason the choice was made to simplify the problem to one of determining the root of the chord. It is noted that information is lost in this process, both in the expressive power of the output and the modality of the input (a $C\ major$ and a $C\ minor$ will now appear the same). This trade off was deemed necessary to allow for meaningful evaluation.

4.3 Baselines

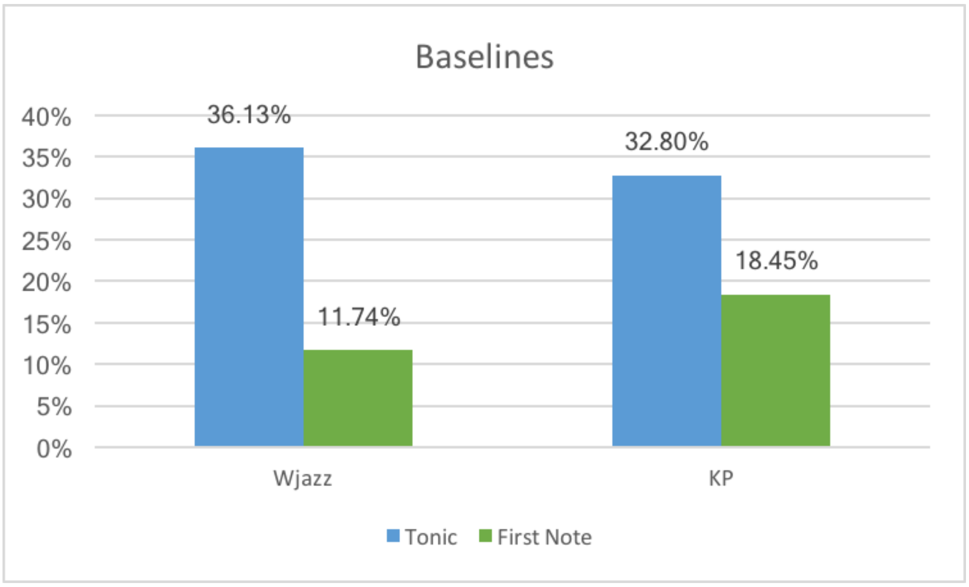
To evaluate the models designed in this project, baseline models were first necessary to define and run on the two datasets. In the previous year's work the baseline that was chosen was simple random chance (i.e. $\frac{1}{K}$ where K is the number of chords). It was decided that this year a stronger baseline would be used to better gauge how much the models are capturing.

Two baselines are proposed. The first is the $Tonic$ model. This model looks at the key the piece is in and guesses the tonic chord of the key. More accurately, this model guesses C when all the pieces are transposed to C.

The $Tonic$ model is a strong baseline as information on the key is, in many cases, very indicative of the chords you can expect to see in the piece. Hence a weaker baseline is also proposed, the $FirstNote$ model, that looks at the first note in every piece and guesses the chord with that note as its root. This model is, perhaps, a more reasonable baseline, as we are giving it no information about the piece of music ahead of time.

In implementing these models, it was important to maintain the one-to-one, repeated mapping between chords and notes used in the main models. If the original, shorter sequence of chords was used then the models would not be comparable.

Interestingly these models give dramatically different results on the two different datasets. The results of running them on Weimar Jazz Database (WJazz) and the KP Corpus are presented below:



This gives valuable insight into the nature of the two datasets and how the pieces they contain differ fundamentally in structure. The first key observation is that the jazz pieces depend much more heavily on key than the classical pieces, the *Tonic* model far out performs the *FirstNote* model on the WJazz data, showing us that 36.13% of the chords in that dataset are the just the tonic chord. This makes the *Tonic* model very difficult to beat.

The *Tonic* model is still the better indicator in the KP corpus but the *First Note* model seems to capture more information than in the WJazz dataset. This makes intuitive sense, as these excerpts are simpler melodically and hence more predictable. The *FirstNote* model performing poorly on the jazz pieces can be attributed to melodies being far more likely to start on the root note than solos.

The inclusion of both of these baselines is, then, important in evaluating the models as they will help us gain an insight into what each model is able to capture from the data.

4.4 Main Results

The method of evaluation for all models was 10-fold cross validation, where the reported accuracies are the means of the accuracies over each fold, and the reported error bars are the standard deviations. This method of evaluation was possible due to the small sizes of the datasets, it would become infeasible, for specifically the LSTM model, if the size of the data was larger.

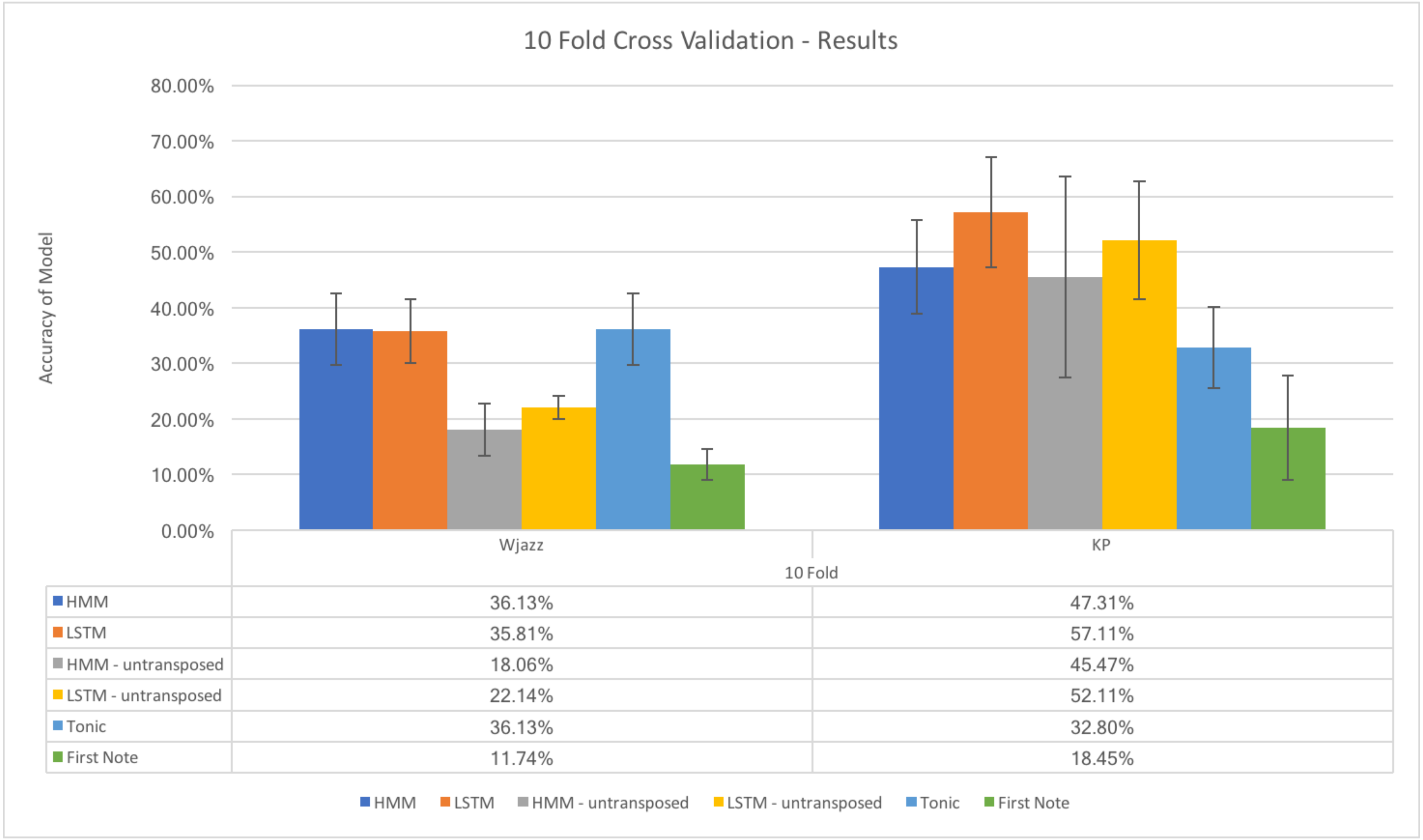
However the small size of the data does have a negative impact on the models. The successful LSTM models of recent years have, in the most part, had large datasets to train on. Hence the small size of these datasets perhaps does not allow the full potential of these models to be displayed.

Despite this, the results obtained from the 10-fold cross validation main experiments are promising. The models were run on the same 10 folds for both datasets. For reproducibility the folds were generated after shuffling the data with `Numpy`'s `permutation` method, with a random seed of 42, and then using the `KFold` index generator from `scikitlearn` [54] for most models. The baselines and the HMM model on the Weimar Jazz Database used the `SequenceKFold` method from `seqlearn` for convenience (with the same random seed).

The LSTM model was trained with 500 epochs, using a 0.4 probability dropout [69] on a 50 cell LSTM layer.

The results of this 10-fold evaluation on both datasets are shown in Figure 4.1. For extra insight into what each model was able to capture, the results of training the models on untransposed data are also shown (i.e. where the songs were not all transposed to the key of *C*). The accuracy values correspond to the mean accuracy over the 10 folds, and the error bars shown are the standard deviations.

Figure 4.1: Main Results



	Wjazz	KP
🟦 HMM	36.13%	47.31%
🟧 LSTM	35.81%	57.11%
⬜ HMM - untransposed	18.06%	45.47%
🟨 LSTM - untransposed	22.14%	52.11%
🟦 Tonic	36.13%	32.80%
🟩 First Note	11.74%	18.45%

The first thing to note from these results is that there was not a model that performed better than the *Tonic* baseline on the WJazz data. In fact, on inspection, this is exactly what the HMM model managed to capture. The LSTM approximated this approach as well, with a little more variation in the output, but nothing that was able to beat the guessing the *Tonic* chord. This confirms the suspicion of the previous year's report, that this task is simply infeasible on the WJazz data.

What we see from the KP results, though, is that the problem with the jazz excerpts is not that there was not enough data, but that the problem is intrinsically too difficult to accomplish with the solo transcripts. This is clear from the fact that there are roughly $\frac{1}{6}$ as many excerpts in the KP corpus, but the models were able to perform significantly better than the baselines.

The smaller size of the KP corpus can be seen in the standard deviations of the accuracies. The performance was clearly very dependent on the particular excerpts in each fold. These would be expected to decrease as the data size increases.

From the KP results we can see that transposing the data to the same key causes a very slight improvement to the model, much smaller than with the Wjazz results. This can be understood from the results of the *Tonic* model, where it is the only model that performed worse on the KP corpus. Clearly these excerpts do not depend on the key as heavily as the jazz pieces. The improvement transposing causes can be understood as the models learning relative relationships between, say, the tonic chord and the dominant fifth chord, rather than the absolute relationship between $C\ major$ and $G\ major$.

As expected the LSTM model out performs the HMM model on the KP corpus, as it is able to learn to capture what the HMM explicitly models, the Markov chain of chords emitting notes, while retaining the ability to model long-term dependencies. Realistically this LSTM has not learned a grammar of chord transitions, but has instead relaxed the first order Markov assumption of the HMM and is able to capture long term relationships between notes. A similar improvement in performance may be noticed if an HMM with a second or third order Markov assumption (c.f. n-gram models) was used.

Chapter 5

Conclusion

5.1 Discussion

This report has presented two models of stochastic, automatic chord labelling for melodic sequences of symbolic notes. The models take into account only the pitch classes and ordering of these notes, and from this they predict the roots of the underlying chord progression.

The results have confirmed one of the suspicions of the previous year's work, that the Weimar Jazz Database is not a suitable dataset for this task. This can now be attributed to the fact that it is composed of 'noisy' jazz improvisations, as opposed to the size of the dataset.

The results of the two models on the KP corpus have outperformed the chosen baselines. Notably the stronger *Tonic* baseline, that receives information about the key of the song, is outperformed by both models even when they do not receive this key information (the untransposed models). This shows that, even with a small dataset, these models are able to capture some of the harmonic structure of the melody.

5.2 Future Work

The KP corpus is still not a satisfactory dataset for the evaluation of these models due to it's very small size. Due to this, the conclusions drawn from the results are dependent on the assumption that the mean accuracy seen is representative of the larger population of classical songs. For further work to be done in this field a significantly larger corpus of annotated symbolic melodies needs to be compiled.

One method to solve this problem could be to take advantage of recent advances in generative adversarial networks [22]. These models train a generative and a discriminative model in tandem, each improving the other. A system of this form could be trained to generate melodies from chord sequences as well as providing a model to label the chords of real melodies.

The HMM model presented is a generative model of melody generation from chords. Hence the configuration presented here could be used in the generation of simple melodic sequences. The model could also be augmented by language model, such as a grammar of chord progressions (c.f. the one proposed in [70]) or by a simple n-gram model.

The generative HMM model could also be augmented by an LSTM model similar to the one proposed here, but taking chord sequence as inputs and estimating their probabilities.

It is hoped that this field can benefit from future advances in machine learning, but to do so it is necessary for a copyright-free, annotated, and widely available corpus of a significant size to be compiled to stimulate research.

Bibliography

[1] ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., CORRADO, G. S., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., GOODFELLOW, I., HARP, A., IRVING, G., ISARD, M., JIA, Y., JOZEFOWICZ, R., KAISER, L., KUDLUR, M., LEVENBERG, J., MANÉ, D., MONGA, R., MOORE, S., MURRAY, D., OLAH, C., SCHUSTER, M., SHLENS, J., STEINER, B., SUTSKEVER, I., TALWAR, K., TUCKER, P., VANHOUCKE, V., VASUDEVAN, V., VIÉGAS, F., VINYALS, O., WARDEN, P., WATTENBERG, M., WICKE, M., YU, Y., AND ZHENG, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] ABESSER, J., FRIELER, K., PFLEIDERER, M., AND ZADDACH, W.-G. Introducing the jazzomat project-jazz solo analysis using music information retrieval methods. In *Proceedings of the 10th International Symposium on Computer Music Multidisciplinary Research (CMMR) Sound, Music and Motion, Marseille, France* (2013), pp. 187–192.

[3] ALLAN, M., AND WILLIAMS, C. K. Harmonising chorales by probabilistic inference. *Advances in neural information processing systems 17* (2005), 25–32.

[4] BILES, J. A. Genjam: A genetic algorithm for generating jazz solos. In *ICMC* (1994), vol. 94, pp. 131–137.

[5] BUITINCK, L. seqlearn. `http://larsmans.github.io/seqlearn/`, 2013.

[6] BURGOYNE, J. A., WILD, J., AND FUJINAGA, I. An expert ground truth set for audio chord recognition and music analysis. In *ISMIR* (2011), vol. 11, pp. 633–638.

[7] CHOLLET, F. keras. `https://github.com/fchollet/keras`, 2015.

[8] CHUAN, C.-H., AND CHEW, E. A hybrid system for automatic generation of style-specific accompaniment. In *Proceedings of the 4th International Joint Workshop on Computational Creativity* (2007), Citeseer, pp. 57–64.

[9] COHEN, C. Music: A universal language. *Music and conflict transformation. Harmonies and dissonances in geopolitics* (2008), 26–39.

[10] COHEN, L. *Songs from a Room.* Song BMG Music Entertainment, 1969.

[11] CONKLIN, D. Music generation from statistical models. In *Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences* (2003), Citeseer, pp. 30–35.

[12] CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS) 2*, 4 (1989), 303–314.

[13] DENG, L., SELTZER, M. L., YU, D., ACERO, A., MOHAMED, A.-R., AND HINTON, G. E. Binary coding of speech spectrograms using a deep auto-encoder. In *Interspeech* (2010), Citeseer, pp. 1692–1695.

[14] DI GIORGI, B., ZANONI, M., SARTI, A., AND TUBARO, S. Automatic chord recognition based on the probabilistic modeling of diatonic modal harmony. In *Multidimensional Systems (nDS), 2013. Proceedings of the 8th International Workshop on* (Sept 2013), pp. 1–6.

[15] DUŠEK, O., AND JURČÍČEK, F. Sequence-to-sequence generation for spoken dialogue via deep syntax trees and strings. *arXiv preprint arXiv:1606.05491* (2016).

[16] DYLAN, B. *Blood on the Tracks*. Ram's Horn music, 1975.

[17] EBCIOĞLU, K. An expert system for harmonizing chorales in the style of js bach. *The Journal of Logic Programming 8*, 1-2 (1990), 145–185.

[18] ECK, D., AND LAPALME, J. Learning musical structure directly from sequences of music. *University of Montreal, Department of Computer Science, CP 6128* (2008).

[19] FORNEY, G. D. The viterbi algorithm. *Proceedings of the IEEE 61*, 3 (1973), 268–278.

[20] FORTE, A. *Tonal harmony in concept and practice*. Holt, Rinehart and Winston, 1962.

[21] FORTE, A. Syntax-based analytic reading of musical scores.

[22] GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A., AND BENGIO, Y. Generative adversarial nets. In *Advances in neural information processing systems* (2014), pp. 2672–2680.

[23] GRANROTH-WILDING, M., AND STEEDMAN, M. A robust parser-interpreter for jazz chord sequences. *Journal of New Music Research 43*, 4 (2014), 355–374.

[24] GREFF, K., SRIVASTAVA, R. K., KOUTNÍK, J., STEUNEBRINK, B. R., AND SCHMIDHUBER, J. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems* (2016).

[25] HAN, S., KANG, J., MAO, H., HU, Y., LI, X., LI, Y., XIE, D., LUO, H., YAO, S., WANG, Y., ET AL. Ese: Efficient speech recognition engine with sparse lstm

on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (2017), ACM, pp. 75–84.

[26] HEBB, D. O. The first stage of perception: Growth of the assembly. *The Organization of Behavior* (1949), 60–78.

[27] HINTON, G. E., AND SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. *science 313*, 5786 (2006), 504–507.

[28] HOCHREITER, S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6*, 02 (1998), 107–116.

[29] HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[30] HORNIK, K., STINCHCOMBE, M., AND WHITE, H. Multilayer feedforward networks are universal approximators. *Neural networks 2*, 5 (1989), 359–366.

[31] IÑESTA, J. M., CONKLIN, D., AND RAMÍREZ, R. Machine learning and music generation, 2016.

[32] KOSTKA, S., AND PAYNE, D. Tonal harmony, with an introduction to twentieth-century music . me graw-hill companies, 2009.

[33] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[34] KRÜGER, J., AND WESTERMANN, R. Linear algebra operators for gpu implementation of numerical algorithms. In *ACM Transactions on Graphics (TOG)* (2003), vol. 22, ACM, pp. 908–916.

[35] KRUMHANSL, C. L., BHARUCHA, J. J., AND KESSLER, E. J. Perceived harmonic structure of chords in three related musical keys. *Journal of Experimental Psychology: Human Perception and Performance 8*, 1 (1982), 24.

[36] KUPIEC, J. Robust part-of-speech tagging using a hidden markov model. *Computer Speech & Language 6*, 3 (1992), 225–242.

[37] LADEN, B., AND KEEFE, D. H. The representation of pitch in a neural net model of chord classification. *Computer Music Journal 13*, 4 (1989), 12–26.

[38] LEBEDEV, S. hmmlearn. `https://github.com/hmmlearn/hmmlearn`, Feb 2015.

[39] LEE, H.-R., AND JANG, J. R. i-ring: A system for humming transcription and chord generation. In *Multimedia and Expo, 2004. ICME'04. 2004 IEEE International Conference on* (2004), vol. 2, IEEE, pp. 1031–1034.

[40] LEES, R. B., AND CHOMSKY, N. Syntactic structures. *Language 33*, 3 Part 1 (1957), 375–408.

[41] LUONG, M.-T., LE, Q. V., SUTSKEVER, I., VINYALS, O., AND KAISER, L. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114* (2015).

[42] MAXWELL, H. J. An expert system for harmonizing analysis of tonal music, understanding music with ai: perspectives on music cognition, 1992.

[43] MCAFEE, F. *Automatic Harmonic Analysis of Jazz Melodies*. 2016.

[44] MINSKY, M., AND PAPERT, S. Perceptrons.

[45] MOON, T. K. The expectation-maximization algorithm. *IEEE Signal processing magazine 13*, 6 (1996), 47–60.

[46] MÜLLER, M., KONZ, V., BOGLER, W., AND ARIFI-MÜLLER, V. Saarland music data (SMD).

[47] NIERHAUS, G. *Algorithmic composition: paradigms of automated music generation*. Springer Science & Business Media, 2009.

[48] NUMPY. Numpy. `http://www.numpy.org/`, 2005.

[49] OH, K.-S., AND JUNG, K. Gpu implementation of neural networks. *Pattern Recognition 37*, 6 (2004), 1311–1314.

[50] OLAH, C. Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, Aug 2015.

[51] OUDRE, L., FÉVOTTE, C., AND GRENIER, Y. Probabilistic template-based chord recognition. *Audio, Speech, and Language Processing, IEEE Transactions on 19*, 8 (2011), 2249–2259.

[52] OUDRE, L., GRENIER, Y., AND FÉVOTTE, C. Template-based chord recognition: Influence of the chord types. In *ISMIR* (2009), pp. 153–158.

[53] PARDO, B., AND BIRMINGHAM, W. P. Algorithms for chordal analysis. *Computer Music Journal 26*, 2 (2002), 27–49.

[54] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., AND DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[55] PICHOTTA, K., AND MOONEY, R. J. Using sentence-level lstm language models for script inference. *arXiv preprint arXiv:1604.02993* (2016).

[56] PISTON, W. *Harmony*. Norton, 1948.

[57] RAGAN-KELLEY, M., PEREZ, F., GRANGER, B., KLUYVER, T., IVANOV, P., FREDERIC, J., AND BUSSONIER, M. The jupyter/ipython architecture: a unified view of computational research, from interactive exploration to communication and publication. In *AGU Fall Meeting Abstracts* (2014), vol. 1, p. 07.

[58] REGENER, E. *Pitch notation and equal temperament: A formal study*, vol. 6. University of California Press, 1973.

[59] ROHRMEIER, M. A generative grammar approach to diatonic harmonic structure. In *Proceedings of the 4th sound and music computing conference* (2007), pp. 97–100.

[60] ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review 65*, 6 (1958), 386.

[61] RUSSELL, S. J., AND NORVIG, P. Artificial intelligence: a modern approach (international edition).

[62] RYYNÄNEN, M. P., AND KLAPURI, A. P. Automatic transcription of melody, bass line, and chords in polyphonic music. *Computer Music Journal 32*, 3 (2008), 72–86.

[63] SCHENKER, H. *Harmony*, vol. 1. University of Chicago Press, 1979.

[64] SCHMIDHUBER, J., GERS, F., AND ECK, D. Learning nonregular languages: A comparison of simple recurrent networks and lstm. *Neural Computation 14*, 9 (2002), 2039–2041.

[65] SHORE, J., AND JOHNSON, R. Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *IEEE Transactions on information theory 26*, 1 (1980), 26–37.

[66] SIMON, I., MORRIS, D., AND BASU, S. Mysong: automatic accompaniment generation for vocal melodies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2008), ACM, pp. 725–734.

[67] SMAILL, A., WIGGINS, G., AND HARRIS, M. Hierarchical music representation for composition and analysis. *Computers and the Humanities 27*, 1 (1993), 7–17.

[68] SMOLIAR, S. W. A computer aid for schenkerian analysis. In *Proceedings of the 1979 annual conference* (1979), ACM, pp. 110–115.

[69] SRIVASTAVA, N., HINTON, G. E., KRIZHEVSKY, A., SUTSKEVER, I., AND SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*, 1 (2014), 1929–1958.

[70] STEEDMAN, M. J. A generative grammar for jazz chord sequences. *Music Perception: An Interdisciplinary Journal 2*, 1 (1984), 52–77.

[71] SUNDERMEYER, M., SCHLÜTER, R., AND NEY, H. Lstm neural networks for language modeling. In *Interspeech* (2012), pp. 194–197.

[72] SUTSKEVER, I., VINYALS, O., AND LE, Q. V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (2014), pp. 3104–3112.

[73] TIELEMAN, T., AND HINTON, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning 4*, 2 (2012).

[74] TURETSKY, R. J., AND ELLIS, D. P. Ground-truth transcriptions of real music from force-aligned midi syntheses.

[75] UZIELA, K., HURTADO, D. M., SHU, N., WALLNER, B., AND ELOFSSON, A. Proq3d: Improved model quality assessments using deep learning. *Bioinformatics* (2017), btw819.

[76] VINCENT, P., LAROCHELLE, H., BENGIO, Y., AND MANZAGOL, P.-A. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning* (2008), ACM, pp. 1096–1103.

[77] WALKER, J. Midicsv: Convert midi file to and from csv. `http://www.fourmilab.ch/webtools/midicsv/`, Jan 2008.

[78] WEN, T.-H., GASIC, M., MRKSIC, N., ROJAS-BARAHONA, L. M., SU, P.-H., VANDYKE, D., AND YOUNG, S. Multi-domain neural network language generation for spoken dialogue systems. *arXiv preprint arXiv:1603.01232* (2016).

[79] WERBOS, P. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, 1974.

[80] WERBOS, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE 78*, 10 (1990), 1550–1560.

[81] WHORLEY, R. P., AND CONKLIN, D. Music generation from statistical models of harmony. *Journal of New Music Research 45*, 2 (2016), 160–183.

[82] WIDMER, G. Qualitative perception modeling and intelligent musical learning. *Computer Music Journal 16*, 2 (1992), 51–68.

[83] WINOGRAD, T. Linguistics and the computer analysis of tonal harmony. *journal of Music Theory 12*, 1 (1968), 2–49.

[84] ZHANG, Y., JIN, R., AND ZHOU, Z.-H. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics 1*, 1-4 (2010), 43–52.