Utrecht University

MSc Thesis in

Technical Artificial Intelligence

Teaching neural networks to play the piano

By Sam van Herwaarden

Thesis number ICA-1355678
March 7, 2014

Supervisors:
Dr. Maarten Grachten[1]
Dr. Bas de Haas[2]
Dr. Frans Wiering[2]

[1]*Österreichisches Forschungsinstitut für Artificial Intelligence*
[2]*Utrecht University*

Abstract

Expressive accentuation in music is a phenomenon that is hard to predict formally, given only a musical score. In this thesis, we propose methods to predict expressive parameters of piano music: dynamics (loudness), tempo and timing. Two different models are applied, one parsing music on a note-by-note basis, and the other parsing the music beat-by-beat. Unsupervised feature learning with sparse RBMs is first used on more than 6000 musical pieces to find recurring patterns (features) in the input score, after which these are correlated to expressive parameters using a supervised learning approach. For dynamics, the system achieves a better R^2 score than the current state of the art on this dataset. The system also exhibits characteristics that indicate that it has the capacity to learn patterns specific to certain genres/musical styles. For tempo and timing, the model has almost no predictive value. A number of suggestions for developing the system further are made.

Preface

This thesis is the product of a research project for completion of the MSc degree in Computer Science (Technical Artificial Intelligence) at Utrecht University. I did a large part of the work for this project in the form of a research internship at the Austrian Research Institute for Artificial Intelligence (OFAI) in Vienna, where music research makes up a very significant part of their activities.

The goal of this project was for me to gain a better understanding of both machine learning techniques and the structural and cognitive aspects of music, and to experience artificial intelligence (AI) research in a real world setting. Of course, if possible, the goal was also to contribute to the field of music information research, and perhaps even expand our collective knowledge.

The thesis is targeted at an audience of computer scientists and music researchers, but is also intended to be accessible for anyone with an interest in AI (particularly neural networks and pattern recognition) or musical expression. The relevant background for these fields is summarized in this report, and the contents should be understandable for anybody from a scientific background.

I thoroughly enjoyed this project, and realize that it would not have been possible without the support of a number of people. First of all, I would like to thank Maarten Grachten: I could not have wished for a more dedicated supervisor, and without his knowledge on the subject I would not have gotten anywhere. Many thanks also go to my supervisors in Utrecht, Bas de Haas and Frans Wiering, who provided a lot of valuable guidance particularly while writing this report. I would also like to thank prof. Gerhard Widmer for giving me the opportunity of doing research at the OFAI, and prof. Remco Veltkamp and prof. John-Jules Meyer for taking on the role of examiner. Thanks also go to Jan Schlüter for frequent assistance with the technical part of the project, and of course to my family and friends (both in the Netherlands and in Vienna) for their support during the entire process. Finally, I want to note that this project was funded in part from the Wittgenstein award, and in part from the European project "Learn to Create".

Contents

Chapter 1

Introduction

Whoever listens to music can recognize that music performance is a highly dynamic concept: there is generally a lot of variation in how different notes are played and the character of a piece changes as it progresses. This is in fact a very important component of music that makes it interesting: a mechanical performance that precisely respects the notation in the score is often considered dull, and just as with spoken language, musical language requires intonation to make it interesting. The way in which this intonation is applied depends on many factors (Palmer 1997, Repp 1992) and although performers have significant freedom in their interpretation of a piece, not all interpretations are musically appropriate.

This thesis documents an effort to develop a system that models musical expression and can in fact interpret musical score expressively. To limit the scope and complexity of this effort, we focus on piano music, where most expression is encoded by variations in time and loudness due to the nature of the instrument.

Before describing the system, experiments and results, it is necessary to cover some context and background first. Section 1.1 explains why the subject of expression in musical performances is interesting to study, and what makes it difficult to analyse. Section 1.2 describes the objective of this particular project, and Section 1.3 gives an overview of the structure of the rest of this thesis.

1.1 Problem description

Langner & Goebl (2003) and Widmer et al. (2003) generated visualizations of piano performances by different pianists, where the pianists played the same piece. In their results it can clearly be seen that for a large part, pianists agree on how to place accents – partly, this is to be expected, since composers add score annotations describing how a particular piece of music should be played. However, using only score annotations to explain this common component is not enough: Grachten & Widmer (2011) modeled expressive parameters on the basis of score annotations, but were able to explain only a part of the variation in loudness.

These observations suggests that a part of the expression is shared among pianists, even though it is not explicitly denoted in the score. Part of this can be attributed to different styles: for example genres tend to display consistencies in terms of how music in that genre is performed, and there are commonalities through tradition and training. Also, Palmer (1997) claims that beside commonalities due to score annotations and differences due to performer style, *"there are also strong commonalities across performances that reflect cognitive functions of grouping, unit identification, thematic abstraction, elaboration, and hierarchical nesting."*

Although apparently pianists tend to agree on the accents that a piece implies, it is unclear which features of the piece affect this, and how. Researchers such as Todd (1989), Friberg et al. (1991), and Bisesi & Parncutt (2010) have attempted to use musicological formalisms to relate structure to expression. However, they only partially succeeded, and the formalisms were defined and tested by hand. There are also other possibilities: Widmer et al. (2003) give an overview of several efforts to understand musical expression using artificial intelligence (AI) techniques. Still, even successful partial models of expression, such as those in another paper by Widmer (2003), depend strongly on hand-designed score features.

There are some clear advantages of hand-designed features: they explicitly encode our understanding of musical structure, and are therefore justifiable and reliable. However, there are also several limitations to the approach:

- Hand-designed features inherently focus on knowledge that is already available, and are therefore unlikely to result in novel insights.
- Because there are large variations in expression, for example between genres, hand-designed features are very specific to a certain situation (i.e. we might need different features and rules for ballads and waltzes, even if they are from the same composer in the same period). In part, features and rules describing one style can possibly be re-used for another, but describing a different style in any case will require a large amount of hand-work.
- It is questionable whether the full range of musical structure and expres-

> sion can be captured in simple rules at all. In this sense, music seems to be very similar to language: there are many irregularities, and highly complex structures and relations. Although previous studies have been able to describe expression in performances to some extent, success is limited.

Using different techniques it might be possible to (partly) overcome these limitations, for example by using systems that are not based on hand-designed features. In fact there are AI techniques with which this is possible: unsupervised feature learning is a growing field in AI, where recurring patterns (features) in data are automatically isolated by machine learning systems. Input can then later be tested to see which features occur in a specific sample, and these features can then be related to other properties of the input data.

This is what we aim to develop in this thesis: techniques that autonomously discover features in musical score, in order to relate these features to parameters of musical expression. By quantifying these relations, it is then possible to transform the features and correlations into a system that predicts expression, which is in essence similar to the rule-based systems but with potentially more modeling power and less required hand-work.

1.2 Research objective

The research question this study should answer is the following:

> *To what extent can expressive parameters in music be predicted from features learned in an unsupervised fashion, and do these features give insight in underlying musicological concepts?*

An initial step in this direction has been taken by Grachten & Krebs (2014), modelling dynamics. They compared several feature learning techniques, of which Restricted Boltzmann Machines (RBMs, Hinton & Sejnowski 1986) performed best. We hypothesize that this approach can be improved and/or expanded to predict more expressive parameters. Whereas Grachten & Krebs (2014) focused on comparing different techniques and configurations, this thesis focuses on applying RBMs in a broader range of contexts: experimenting with different musical styles, and different expressive parameters, while adapting the RBM setup for this particular application. The next chapter describes in more detail what an RBM does and how it works.

This project has several objectives. The first objective is to develop a model that can predict musical expression given only a musical score – this can be used to *render* performances for, for example, MIDI data. We can test this model by comparing predicted performances to actual (human) performances. The second objective is to use this model to *explain* expression in existing performances: in other words we want to understand why accents are placed the way they are. This implies that to some extent it should be possible to observe the inner workings of the model.

1.3 Structure of this thesis

This project is rooted both in music theory and in machine learning, and as such also builds heavily on concepts from both fields. Therefore, both understanding the applied models and interpreting the results require some background knowledge. Chapter 2 covers these basics, where the aim is to be in-depth enough that the rest of the thesis is understandable, while avoiding unnecessary technical details. Chapter 3 describes the overall setup that is used for all experiments. Various experiments have been performed, some relating to expression in dynamics/loudness, and others relating to tempo and timing – these are described in Chapters 4 and 5, respectively. Finally, Chapter 6 summarizes the main observations and suggests approaches for possible follow-up work.

Chapter 2

Background

This chapter serves to get the reader up-to-speed with regard to background knowledge required to understand the rest of this thesis. Section 2.1 covers the basics of music theory, whereas Section 2.2 gives an introduction to the main concepts of machine learning and neural networks. Readers with a strong background in either of these fields can skip the corresponding sections. Section 2.3 summarises some work that has been done before in this subject, both from a musicological perspective and applying machine learning techniques.

2.1 Basics of music theory

This section outlines some basic principles of music theory, which play an important role in the design of our system and the interpretation of results. Only concepts that are relevant to this project are discussed: the role of timbre (audible structure of a sound) and interaction of instruments are important topics in music theory, but have little relevance for this thesis – we only consider solo piano music, and do not take into account timbre changes (pedaling). The following topics are covered: harmony (2.1.1), rhythm (2.1.2), and form (2.1.3).[1]

2.1.1 Harmony and pitch

In its bare form, Western music generally consists of notes, each of which has a pitch, a duration, and a time at which it sounds. Pitch is the perceived fundamental frequency of the tone corresponding to the note, where (young) people can hear frequencies from approximately 20Hz to 20,000Hz. At any one point in time in a musical piece, normally not the full spectrum is used, but only a set of frequencies at fixed distances from each other.

When a certain pitch has exactly a power of 2 times the frequency of another pitch, these are considered to be in the same pitch class, and they are

[1]For a more thorough introduction to music theory the Open Yale course *Listening to Music* is recommended, which is currently available at `http://oyc.yale.edu/music/musi-112`.

perceptually closely related. Pitch classes are referred to with the letters A to G, possibly in combination with a flat ♭ or sharp # indication. Sometimes different pitch classes are also referred to by do, re, mi, fa, sol, ... If a pitch has exactly twice the frequency of another pitch, these are said to be one octave apart. With this fact we can quickly calculate the number of octaves in human hearing: $\log_2\left(\frac{20{,}000}{20}\right) \approx 10$.

Within one octave, there are 12 pitch classes, each a semi-tone (i.e. 1/12th of an octave) apart. When tones of different pitches sound simultaneously, there is a strong relation between how far apart they are within the octave, and how pleasant they sound together: combinations of tones with particular intervals sound significantly better than others. People are very sensitive to this: although most people do not have absolute pitch (the ability to exactly tell the pitch of a certain tone), they are sensitive to relative pitch, especially for notes playing at the same time, and people have a strong sense of which tones 'belong' together (Steinbeis, Koelsch & Sloboda 2006).

One particular interval that sounds well is when notes are 7 semi-tones apart. We can quickly calculate what this says about the ratio of the frequencies of the two notes: one has $2^{7/12} \approx 3/2$ times the frequency of the other. This illustrates the fact that how well tones combine (i.e. how pleasant they sound together) is very mathematical, and it is related to the characteristics of the resulting waveform when waveforms with different frequencies are combined. When we take tones that are closer together, but not of the same pitch class, the combination often sounds less pleasant, for example when combining frequencies only 1 semitone apart.

Because not all combinations sound pleasant, generally not all pitch classes are used, and definitely not all at the same time. There are certain trends and patterns in which combinations are used together, and how this changes over time. The analysis of these patterns is harmonic analysis. The following sections describe a number of parameters that say something about the harmony within a piece. A good book that covers harmony in more detail was written by Piston, DeVoto & Jannery (1978).

Musical scales As was mentioned previously, normally not all pitch classes, but a subset of them are used in a piece of music. However, there are many different possible subsets that can be chosen, and whereas the consonance or dissonance of different tones is a very physical property, the choice between these different subsets is a much more cultural phenomenon.

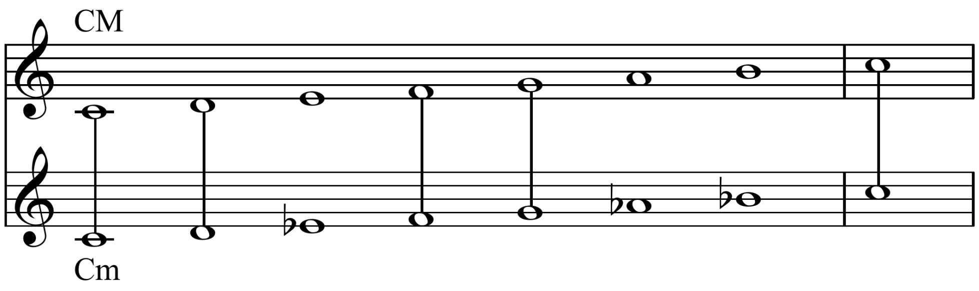


Figure 2.1 Comparison of the major (top) and minor (bottom) scales. Both are illustrated with C as root.

A number of subsets are commonly used, and they are referred to as musical scales. They build upon a root pitch (which can be any of the pitch classes), and are defined as a set of intervals from the root which to include. The most common scales in Western music are major and minor. Both are heptatonic scales (meaning that 7 pitch classes are included), but they differ in the distances between the included pitch classes and the root (see Figure 2.1). The applied scale has a strong effect on a listener's interpretation: culturally, over the centuries Western music has evolved in such a way that happy music is generally written in a major scale, while sad music (for example funeral music) is written in a minor scale. As soon as a person starts listening to music, this connotation also has a strong effect on the way the music is experienced, even if a person does not know about scales.

An important fact to stress here is the cultural component: Jewish music is often also played in minor scale, and can be very happy nonetheless. Variations also occur between musical styles, for example blues and jazz use a different scale. In Asian music often very different scales where 5 instead of 7 of the pitch classes are used, so-called pentatonic scales. These are also often applied in impressionist music and in blues.

Key The key of a piece of music is the combination of the scale and the corresponding root note which form the basis for the music. This root tone is also know as the tonic, and has the property that listeners tend to 'want' the music and its melody to go to the tonic – until the tonic sounds, a musical phrase does not feel as if it has finished. Other important tones are the dominant (7 semi-tones up from the tonic) and the subdominant (7 semi-tones down, with a frequency $2^{-\frac{7}{12}} \approx \frac{2}{3}$ times that of the tonic). Both of these sound consonant in combination with the tonic.

Again, because most people do not have absolute pitch, the precise key that is applied is unlikely to be very significant. However, it defines to a large extent the role that certain tones play in the music – it might be useful to know which tones play which roles, and for this some knowledge about the key could be useful.

Tone sequences When consecutive tones are decreasing in pitch, this gives another effect than tones that are increasing in pitch – the latter is really more 'uplifting'. So-called melodic sequences are also often applied, where a short pattern in music is repeated, and the repetitions are shifted up or down in pitch. We will not go into any formal details, just realise that these effects exist.

Chords and chord progressions Whereas scale refers to the collections of pitch classes used over the span of a musical piece, chords are collections of pitch classes that are played at the same time, and these are typically less than a full scale (mostly 3 or 4 notes sounding together). Many different chord types exist, all with their own properties. Factors that influence the effect of a musical piece are which chords are used, in what order they are used, and when the transitions between chords take place (this can be at regular or irregular intervals).

Texture Texture refers to the number of voices playing at the same time, and in what way they relate to each other. In music played by an orchestra, this can relate to the way different instruments play different lines; in our case (only a single piano) this relates to the musical activity at different pitches on the same instrument.

When multiple lines play at the same time, they generally have to adhere to the rules of harmony: the pitches sounding at the same time should fit together and correspond to the intended progression of the harmony of the piece. This is also referred to as 'counterpoint', and contrapuntal motion refers to the change in the pitch distance between multiple lines playing at the same time.

There are three main classifications with regard to texture: monophony, homophony, and polyphony. Monophony refers to a single voice (one tone sounding at a time), homophony refers to multiple voices playing the same melody but with different reference pitches, and polyphony refers to multiple voices playing different melodies. When listening to polyphonic music, for people it is relatively easy to filter out the different melodies and focus on them separately.

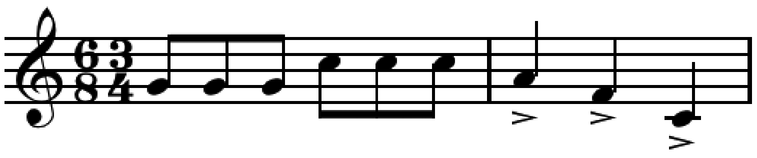


Figure 2.2 Transition from 6/8 to 3/4 (from 'America' in the musical West Side Story, composed by Leonard Bernstein)

Dissonance The effect of dissonance was mentioned before as a consequence of how waveforms combine, but it may have sounded as though the fact that notes do not sound pleasant together means that their combination should be avoided. In fact, this is not the case: dissonance is often applied intentionally, for many different reasons (one reason being that it can enhance the pleasing sound of the consonant tones following it). As was mentioned before, also untrained listeners are very sensitive to this, so it is likely that this is an important factor affecting musical expression.

2.1.2 Rhythm and time

The other important domain in music next to pitch, is time. In fact, just as whole musical genres can use different harmonic patterns, they can also be built upon different types of rhythms. Some notable examples include reggae and jazz, both of which are mentioned later in this section. A number of concepts relate to rhythm and time:

Meter Tones in music tend to form natural groupings over time: often short 'phrases' of music are repeated (possibly as variations), and there tends to be a certain regularity to which tones are accentuated, based on their position in the rhythm. The natural groupings often span a fixed length in musical time (a particular number of beats).

Meter denotes the number of notes, and the corresponding note duration, that make up one such grouping, and these groupings are referred to as measures, for which the boundaries are indicated by vertical bars in the score. This is illustrated in Figure 2.2: it displays a short musical segment of two measures. The vertical line between the notes indicates the transition from one measure to the next, and both measures have a duration of 3 quarter-notes: in the left measure there are 6 eighth-notes, and in the right 3 quarter-notes (which add up to the same total duration). These durations are denoted with the numbers on the left, for the first measure it is 6/8 and for the second 3/4. The upper number indicates the number of notes, and the lower indicates the duration of these notes. These indications in this case correspond exactly to the notes that can be seen in the figure, but note that a meter of 6/8 does not necessarily mean that only eighth notes can exist in a measure – it only means that eighth notes are the primary unit of counting (note durations can be mixed).

The most common meters are 2/4, 4/4, 3/4 and 6/8, but many others are possible. 2/4 and 4/4 are duple meters: their main division is in a factor of 2. 3/4 is a triple meter type, and 6/8 is a compound meter (with both a division in 2, and one in 3). The meter tells a lot about the accents in a piece, because notes in certain positions are stressed (by playing louder for example). The first note in a measure is generally stressed, and other notes are stressed depending on the subdivision. For example a 2/4 meter one would count as ONE two ONE two, a 3/4 meter as ONE two three, and a 6/8 meter as ONE two three FOUR five six. In the figure a transition from 6/8 (implying accents on the first and fourth *eighth*-notes) to 3/4 (with accents on the first, third and fifth *eighth*-notes) is illustrated. A transition between duple and triple meter can also take place – a nice example where this is done is 'Lucy in the sky with diamonds' by The Beatles.

Meter and the way accents are placed over time can be very defining for music. To name only a few examples, waltzes are generally in 3/4, while a lot of modern electronic music is in 4/4 with heavy percussive effects on every quarter-note ('four to the floor'). Reggae, on the other hand, is also in 4/4, but has strong instrumental accents on the second and fourth quarter notes.

Tempo Tempo relates to how fast a piece progresses, and is mostly expressed in BPM (beats per minute). What counts as one beat depends on the meter. Typical tempo's are around 90 BPM but this is highly dependent on the piece, performance, and musical style.

Although it is easy to give a BPM number as an average for a piece (divide the total number of beats by the total time duration), this is not so straightforward for local tempo. The resulting number will depend strongly on how many notes are included in the measurement, and which definition for tempo one chooses (this also affects how tempo can change for example – does it change in a step-wise way, or linearly, or smoothly?). The definition for tempo used in the machine learning system is described in more detail in Chapter 5.

Timing Tempo and timing are generally seen as separate effects: tempo relates to the beat-rate of the music as a whole, timing relates to the offset of tones compared to the beat grid. Still, they both affect the observed on- and offset times, and are therefore hard to separate (especially when considering local tempo variations). In fact, their relationship can be even more complicated: they can influence each other, such that changes in tempo affect the relative timing of notes. For example this effect is strong in jazz (Honing & de Haas 2008). However, for jazz, timing is a major component that defines the genre ('swing' is applied, where notes are played alternatingly shorter and longer). Fortunately, for the piano music studied here, the impact of timing is less severe, but its effect should be considered. The definition for timing used in the machine learning system is described in more detail in Chapter 5.

Triplets Commonly, subdivisions in music over time are related by powers of 2: phrases (musical sentences) are often two, four or 2^n measures long, and notes are also mostly divided in two (half notes, quarter notes, eighth notes, etc). We have seen an exception to this pattern: measures with triple (or compound) meter. At note-level, these exceptions also exist, and are called triplets: 3 triplets have the same duration as 2 notes of a regular duration (for example eighth notes). This affects the rhythmic patterns that can occur in a score, and it also introduces an irregularity in the timing of individual notes: for example, some notes may occur 2/3 of a quarter-note into the measure.

Syncopation When listening to music, people have quite strong expectations with regard to rhythm, affected in part by the choice of meter and the present musical lines. Sometimes, musicians intentionally deviate from the expected rhythm by omitting sounds, or by shifting their timing slightly from the expectation – this can create tension or actually emphasise certain aspects of the rhythm. Its effect clearly relates to a well-developed understanding of rhythm, and is hard to formalise; it would be interesting if a machine learning system builds up some sort of expectation on rhythm, but at the same time this is a rather ambitious goal.

2.1.3 Form

The previous musical characteristics all related to local aspects in music, i.e. the structure at one moment in time, or at most over a few measures. Musical form relates to the global structure of music: groups of measures are ascribed a certain role in view of the entire piece. The best-known musical form is presumably verse-chorus (couplet-refrain), the standard form for pop music. The verse and chorus alternate, where the verse is a little bit different every time it returns, while the chorus stays the same.

In classical music, form is particularly developed: there are many different forms of music intended for different settings (affecting for example the total duration of the piece) or different performance styles (for example showcasing a single instrument, or an orchestra). An in-depth description of different classifications of form is omitted here.

Important building blocks of form are often a theme (a main pattern that returns often in the piece), development of that theme (variations on the main pattern) and transitional sections (leading from one state to another). Clearly, in terms of a feature hierarchy, form is very high up: it builds upon the structures of measures, the structures of groups of measures, and the relations of all these components to each other.

Because form is such a high-level feature, and because of its variability (for example in the duration of different components, or the total duration of a piece) it seems unreasonable for an automated system to fully take into account its implications. A system that would be able to grasp some of these components would require flexibility with regard to the span of a piece that it receives as input, and would require the capability to relate features that are relatively far apart within a piece in terms of time (for example repetitions of melodic lines). However, it is still important to realize that this is a factor that affects structure on the smaller scale as well – for example sometimes a melody is played differently in different parts of the piece, even though in terms of notes it looks identical. This imposes a limit on what can be expected in terms of modeling performance.

2.2 Basics of feature learning and neural networks

To understand the methods used in this study, it is necessary to understand some basics of machine learning. This section elucidates the main principles that were applied, at a sufficient level of detail to grasp the contents of the rest of this thesis. Most technical details are omitted[2]: although they are necessary for the implementation of the system, they do not significantly affect the

[2]The excellent Coursera course *Neural Networks for Machine Learning* covers these thoroughly. It is currently available at `https://www.coursera.org/course/neuralnets`.

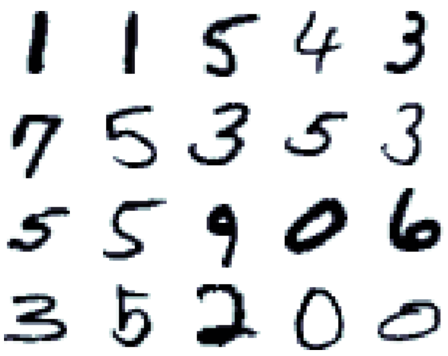


Figure 2.3 Some handwritten digits from the MNIST dataset.

overall design of the system or the interpretation of the results. The descriptions are therefore conceptual, and are often illustrated with examples based on MNIST – this is a standardized hand-written character recognition problem, which is the main problem on which most machine learning systems are tested and benchmarked (it is sometimes referred to as the 'fruit-fly' of machine learning). Figure 2.3 illustrates some samples from the MNIST dataset.

2.2.1 Supervised vs. unsupervised learning

In machine learning, two main styles of learning are distinguished: supervised learning and unsupervised learning. Roughly speaking, a supervised learning system learns by example (it is taught, or supervised, during the process) and an unsupervised learning system needs to find structure in input data all by itself.

A system doing supervised learning on hand-written digits would learn to classify the digits, labeling them as one of the values 0 to 9, while a system doing unsupervised learning would simply look for structure in the data (for example input images sharing common strokes). Unsupervised learning is often used to 'cluster' data, to organize the data into groups when distinction between different kinds of data is desired but it is unclear what this distinction should be exactly.

In this thesis, we actually mix both approaches, and this is also sometimes referred to as semi-supervised learning. We use unsupervised learning to initially find structure in the data, and this information is then used to 'prime' a supervised learning system which relates the features to target values (e.g. the values 0 to 9 for MNIST, or parameters of musical expression in the rest of this

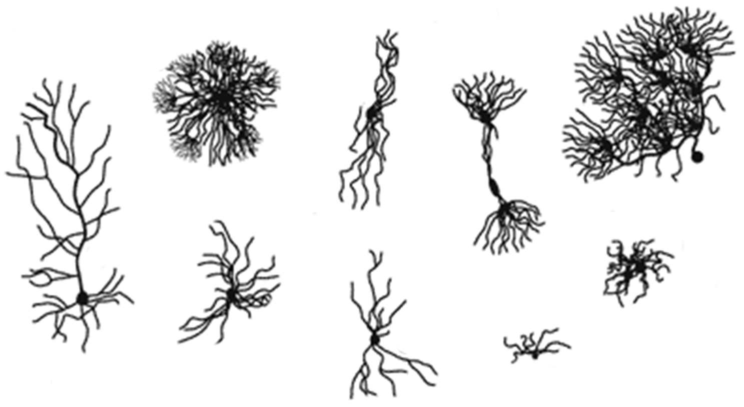


Figure 2.4 Some different neuron types that exist in our brain (based on drawings by Ramón y Cajal).

thesis). Note that the two approaches are not applied at the same time, but in sequence.

2.2.2 Neural networks

Artificial neural networks form a class of machine learning techniques that can be used for both supervised and unsupervised learning. As the name suggests, the method is inspired by mechanisms from our brain. There are different variants of artificial neural networks, the ones we will be using are feed-forward neural networks (FFNNs) and Restricted Boltzmann Machines (RBMs).

Although there are significant parallels between artificial and real-life neural networks, typical artificial networks do not come close to real neural networks in terms of complexity: the human brain contains around 10^{11} neurons forming around 10^{14} connections. This is much more than we can model even on powerful computers. Large artificial neural networks consist of perhaps up to 10 layers of at most a few thousand nodes (artificial neurons) each. Clearly this is still far from the capacity of the human brain in terms of processing power.

Another important difference is that the brain contains around 300 different cell types, with highly distinct characteristics. Some of the variety is illustrated in Figure 2.4. In most artificial neural networks there are less than a handful of different neuron types, if more than one. This is another factor that might allow the processing power of the brain to be far larger than that of artificial neural networks. Still, there is reason to believe that at least in

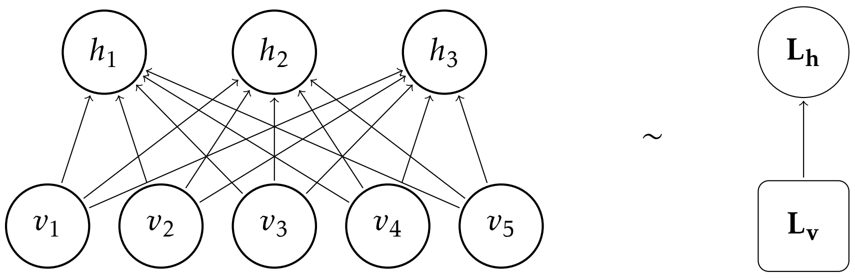


Figure 2.5 A simple neural network with five visible units and three hidden units, in a detailed and an equivalent concise visualization.

the cortex (the outer layer of the brain, which is particularly large in humans) there is a sort of generic learning principle that applies throughout, and which is flexible in terms of what it can learn: in some experiments parts of animal brains were rewired (e.g. connecting inputs for the visual system to the auditory processing system), where one part of the brain took over the function of another (Métin & Frost 1989, Roe et al. 1992). This suggests that at least to some extent the brain uses generic tools for different purposes, meaning that a single learning algorithm and network structure might be sufficient for many applications after all.

The way in which natural and artificial networks store and obtain their knowledge does seem to be quite similar. Lee, Ekanadham & Ng (2007) compare characteristics of artificial neural networks trained on images to those of the visual processing areas (V2) in the brain, and show that there are some clear parallels.

Figure 2.5 illustrates the structure of a very basic neural network (compare the neurons in this network to the neurons in Figure 2.4). There are two layers of nodes (neurons) where each layer is fully connected to the other by edges (synapses). Each of these connections has an associated weight, which indicates how strongly the activation of the bottom node influences the activation of the top node (positively or negatively). In Figure 2.5, the edges are directed, which is indeed the case for FFNNs. In RBMs, the edges are undirected – there is no particular preference for a direction of information transfer, in both directions information is transferred in the same way and with equal connection weights.

Normally in neural networks there is a single input layer, this is the layer for which we set the node activations ourselves. For the handwritten digits, this would imply that every node corresponds to a pixel in the grid, and that node gets the value of the corresponding pixel. Please note that in Figure 2.5, there is no particular structure in the input data: we simply have a sequence of values v_1 to v_5 – when input data is supplied, it is also input as a simple sequence of values (i.e. not as an image). In fact, if we would like we could reverse or even shuffle the ordering of the input, this would not affect the training procedure as long as we reverse/un-shuffle again in the same way when we do any visualization. Later we will visualize features (the connection weights connecting hidden units) as well, and here the same thing happens: they are just sequences of values, but when we visualize them, we transform them into a grid so that we can relate them to the input data.

After the input values are set (typically each unit gets a value of 0 or 1), one can determine the activation of the hidden layers. Each hidden unit receives input from the lower layer, where the contribution of each input unit v_i to hidden unit h_j is scaled by the connection weight w_{ij}. Hidden units also have a bias b_j, which can be seen as describing the prior activation of this unit: a unit can be biased to be on (positive bias), or off (negative bias) more often. The total input y_j for a hidden unit h_j, given input values v_i, connection weights w_{ij} and bias b_j is calculated as follows:

$$y_j = \sum_i (w_{ij} \times v_i) + b_j \tag{2.1}$$

In practice this is implemented as a matrix calculation (calculating many hidden unit inputs from many input samples at the same time).

To calculate the hidden unit *activation* h_j, we apply a transfer function to its input. We use two different transfer functions in this thesis, the logistic function and the linear function, but there are others as well. The logistic function is defined as follows:

$$h_j = \frac{1}{1 + \exp(-y_j)} \tag{2.2}$$

where y_j is the unit input calculated in the previous paragraph. The linear transfer function is much simpler:

$$h_j = y_j \tag{2.3}$$

Note that the logistic function always outputs a value between 0 and 1, while the linear function can output any number. As you can see, the only things we can change here to influence the unit activations are the connection strengths (w_{ij}) and the biases (b_j), and indeed these are the values affected by the learning algorithms, and the values that store the 'knowledge' in a neural network.

2.2.3 Deep networks and pre-training

The network discussed in the previous section was very limited in its modeling power. It might be hard to imagine how simply transforming the input data to a different set of unit activations can be useful. However, if we have a good set of connection weights, there can be a sort of semantic transformation from one layer to the next: in the case of the digit recognition problem, one could imagine that a certain unit activates when there is a loop in the input data, another unit when there is a stroke in the top right, and more such features. This information potentially brings us much closer to understanding the structure of the input, and eventually figuring out which digit it is we are looking at: we can use this network as a basis when we perform supervised learning, as it might be easier to relate a set of strokes to a digit classification than a set of on-off values for a grid of pixels.

The expressive power of the network through transformation becomes more pronounced when we start stacking layers, and in fact in the brain, there seems to be a deep layering where features build upon each other, for example in the visual system (Van Essen et al. 1992). Intuitively, this also makes sense: lower layers could be used for edge detection, layers above that for primitive shapes, followed by layers for more complicated shapes, etc. By stacking multiple layers, we can achieve the same thing in an artificial network. For example, Erhan, Bengio, Courville & Vincent (2009) visualise node activations in deeper networks trained on MNIST data. It can very clearly be seen that nodes in lower layers are responsive to edges, whereas nodes in higher layers are responsive to more complicated combinations of edges and later whole numbers.

Unfortunately, it is hard to train a neural network with many layers: as the learning algorithms proceed through multiple consecutive layers, it becomes less and less clear how changing a weight or bias influences the behavior of the network as a whole. This can make learning very slow, and at the same time this makes it hard to actually find a good configuration for the network.

Still, all is not lost: the layers need not be trained all at the same time: one can build up the network layer by layer. Lower layers can be trained in an unsupervised fashion, where each time the input of a new layer is formed from the activations of the layer below it, which was trained before. In this way, if we're lucky every new layer discovers some regularities in the layer below, 'features'. Not all features might directly be helpful for solving the final task, but it is very likely that features that will be helpful are also found. This way, a feature hierarchy is built up, and when making the step to supervised learning one only has to learn to relate relevant features to the output data.

Different kinds of unsupervised learning can be used to build up a network in this fashion: RBMs (Hinton, Osindero & Teh 2006), denoising autoencoders (Vincent et al. 2008), or contractive autoencoders (Rifai et al. 2011) are commonly used. In this thesis we focus on RBMs.

2.2.4 Restricted Boltzmann Machines

One particular class of artificial neural networks that has been popular for unsupervised learning recently is RBMs. Boltzmann machines have been around for a while already (Hinton & Sejnowski 1986), but have always been particularly hard and slow to train. Only in the last decade some of these challenges were overcome, in part by focusing on the subclass of Restricted Boltzmann Machines where the corresponding graph is bipartite (Carreira-Perpinan & Hinton 2005, Hinton & Salakhutdinov 2006, Hinton 2010). In recent years RBMs have been applied particularly successfully on a varied number of tasks.

As was mentioned before, the edges in RBM graph are undirected: connections go in both directions and weights and transfer functions are symmetric. An important component of RBMs is that they are often used in a generative fashion, where the structure of the network is in fact used to generate samples that could 'fit in' with the training data: essentially one could start with a random activation pattern on the hidden units, and use this to generate realistic input activations. In this thesis this property of RBMs is not used, but the concept is important for understanding the way RBMs learn from their input data.

Figure 2.6 illustrates the structure of an RBM, where $\mathbf{L_v}$ corresponds to the layer of visible units, $\mathbf{L_h}$ to the layer of hidden units and $\mathbf{L_v'}$ to a reconstruction of the visible unit activations from the hidden units. The components of the RBM are also qualitatively visualized:

- The input displays 9 hand-written digits which are fed into the network, these correspond to the visible unit activations (remember that although

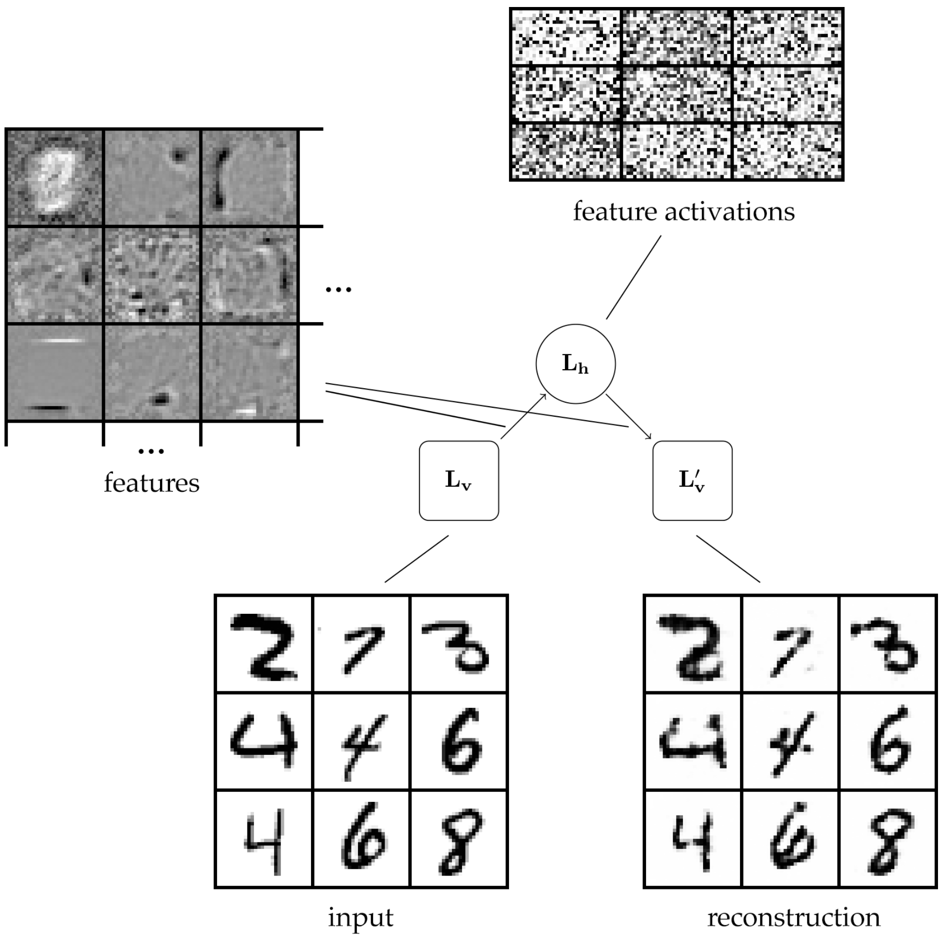


Figure 2.6 A visualization of how the different components of an RBM come together to form a reconstruction. Each input sample (digit) represents activations of the whole input layer (with black for 1 and white for 0). The features corresponding to different hidden units are also illustrated, where a light value corresponds to a positive connection weight, and a dark value to negative (grey means 0). The feature activations illustrate the features that are turned on (black) and off (white) for each of the input samples. The activations are visualized as a grid for compactness, but there is no spatial structure in the activations. The reconstructions are generated by calculating the visible activations *given* the sampled feature activations.

we visualize them as a grid, they are input in the network simply as a sequence of 1's and 0's).

- For each hidden unit, there is a connection weight for each visible unit, and by visualizing the values of these weights, we can visualize the kind of input pattern (the feature) that a hidden unit correlates to. Since for each hidden unit we have as many weights as we have visible units, we can visualize it in the same way as the input (so for 28×28 pixel hand-written digits, we visualize the features as 28×28 pixel grids), and in this way we can give some insight in the knowledge encoded in the neural network. In the figure this is done for 9 hidden units (a small subset of the total number of hidden units). Features give some insight to what the network recognizes, for example the top left unit will activate for images with pixels in the central part of the image turned on.
- The feature activations grid visualizes hidden unit activations for each of the 9 hand-written digits. The feature activations are displayed in a rectangular form, but this is only to save space, there is no spatial structure in the activations. Each rectangle visualizes the activations of the full set of hidden units for the corresponding input samples.
- From the feature activations, it is possible to 'reverse-engineer' the input activations that caused this configuration of the network, simply by propagating unit activations downward (in exactly the same way as was done upward). This is visualized in the reconstruction for each of the input samples. As is clear from the picture, a large amount of the structure of the input data is stored in the hidden unit activations, but the reconstruction is not identical to the input.

When training an RBM, essentially what we do is the following:

1. Set the $\mathbf{L_v}$ activations to the values from the input sample, and determine the corresponding hidden unit activation values in $\mathbf{L_h}$. These are values between 0 and 1.
2. Sample the hidden units so each has an activation of 0 or 1, using the values calculated previously as probabilities. By adding some randomness to the learning in this fashion, we make the network more robust.
3. From the sampled hidden unit activations, determine the corresponding

input unit activations in $\mathbf{L}'_\mathbf{v}$ (the reconstruction).

4. Compare the activations in $\mathbf{L}_\mathbf{v}$ and 1 $\mathbf{L}'_\mathbf{v}$, and tweak the weights and biases in the network such that in the future the reconstruction will be more similar to the input.

The process is started with a random initialization of the network weights, and by iterating these steps many times slowly the weights become more and more suitable for representing the input data. The important point here is that we try to configure the network in such a way, that the hidden unit activations are able to reproduce the visible unit activations accurately using the network weights, in other words the hidden unit activations contain as much of the information that is contained in the input unit activations as possible.

Sometimes steps 1 to 3 are cycled a number of times – the process of iterating the sampling of hidden and visible units is known as contrastive divergence (CD) and when we say we train the network with CDn, this means we iterate the sampling n times. Typically, when you start training the network you use CD1 (computations are fast), and as the network progresses the number of steps is increased to around CD5 or CD10 (this makes learning more accurate).

There is one more key element to this approach: typically the number of (active) hidden units will be significantly smaller than the number of visible units. This causes an information bottleneck: we try to describe all the relevant details of the input data in a smaller number of bits. The hidden units thus 'summarize' the input units in simpler terms than the input representation, where the 'terms' we can use for this summary are the weights corresponding to the different hidden units (the features), which are determined during the training process, in such a way that they are (hopefully) descriptive.

In fact, what happens is very similar to when we apply Principal Component Analysis (PCA): we get a number of basis vectors with which we can more economically describe our input vector. For linear FFNNs this similarity is even stronger, and the learned representation is very similar to PCA (Baldi & Hornik 1989). A downside of PCA is that the transformation is always linear, while especially in deep networks a sequence of non-linear transformations can affect the data in a wide variety of ways. The latter is potentially more powerful.

Something interesting to note about RBMs is that this process, where we reconstruct the observed input from simpler components that we store in our network, also has a parallel in the brain. An important part of our cognitive ability is not so much to just analyze the inputs from our senses, but also to compare what we take in with our senses to what we expect to take in (Kveraga, Ghuman & Bar 2007). Information continually flows up and down through the network, where expectations are used to fine-tune the way we process our input, and discrepancies between our expectations and our senses play an important role when gathering information about our environment.

2.2.5 Hyper-parameters

When training neural networks there are many hyper-parameters the user chooses to configure the learning algorithm. We encountered one example already for RBMs: the number of CD steps. Other examples are the learning rate (how much we change the weights during every training iteration) and the number of hidden units in the network. Unfortunately, the precise configuration of a machine learning system is more of an art than a science: suitable values are typically found through trial and error, and there are typically too many hyper-parameters to optimize the training procedure in every sense.

Bergstra et al. (2011) suggest some approaches for automating the process of hyper-parameter selection, but these add an additional level of complexity to the machine learning system. Hinton (2010) summarizes some 'tricks of the trade' for training RBMs successfully, and the approach in this thesis is mostly based on the observations made in his article.

2.2.6 N-grams

In some cases, we don't so much want to understand structure in separate pieces of input data, but rather relations between different pieces of input data, for example in sequences. For music and language, this is clearly important: the elements separately do not convey much meaning, meaning is conveyed by the way these elements are put together.

The term N-gram comes from computational linguistics and is used to describe a set of N items in sequence. If we take the set of words "The dog bit the cat" this is a 5-gram, and "The dog bit" or "bit the cat" would be examples of 3-grams. Even though we lose a lot of context (if we see "bit the cat" we have no clue who or what actually bit the cat), N-grams are useful because there are often many repetitive patterns, for example if you know that the first two words are "The dog" it is quite likely that the third word will be a verb of some kind.

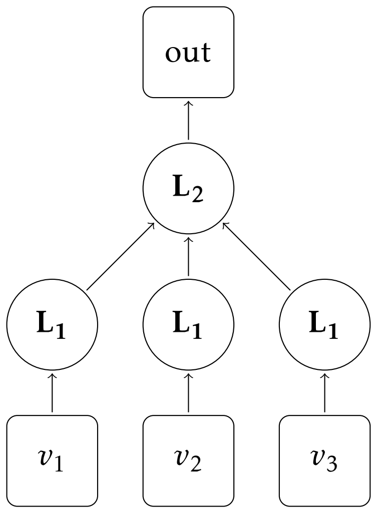


Figure 2.7 An example architecture using a 3-gram (tri-gram) as input.

Originally these models would be used in a relatively straight-forward way, calculating probabilities: one can count how often a third word occurs given the first two words, and from that make an estimate which word is likely to come next. However, in this approach the system does not really learn things about structure, just about co-occurrences. Bengio et al. (2006) took a similar approach but combined it with a neural network: a neural network is trained with as input a representation of the first set of items in the N-gram, where the target output is the last item. In this way the network learns to predict the next word in a sequence.

An even more powerful extension of this is when the N-gram does not consist of the original representations of each word, but of a representation that already contains some information about the word. This approach is illustrated in Figure 2.7. One can see that the input is an N-gram of items, in the sequence in which they naturally occur. Each of these items is then transformed to an alternative representation (in $\mathbf{L}_1$) which contains information about the input item, and finally a top layer ($\mathbf{L}_2$) learns to represent patterns in the sequences that it observes.

Note that each of the input items is transformed in the same way: each of the $\mathbf{L}_1$ layers is built from the same weights and biases, but has different hidden unit activations because of the different input activations. $\mathbf{L}_2$ then learns patterns in the sequences of $\mathbf{L}_1$ activations. This transformation of the sequential input data can be used for predictions on the input, such as the next word. Later you will see that we use this approach to relate sequences in music to the tempo changes in that part of the score.

2.2.7 Sparsity and selectivity

In the past few sections, we talked about how neural networks can encode information about input data. However, we only discussed this in broad terms, and in fact there are different ways in which hidden units can encode for features of the input data. For example, in PCA the bases that we use to describe our data have an ordering in how much of the data they explain: the first principal component generally explains a large part of the information in an input sample, the second a bit less, and so on. In neural networks, typically the information about the input data is spread more evenly over the different hidden units.

Still, there can be different patterns of hidden unit activations. There can be certain neural codings where almost all hidden units are used to some extent to describe the input data (so let's say if we have four hidden units, that for a typical sample their values would be 0.7, 0.3, 0.4 and 0.8) while in other codings each input sample is described by only a fraction of the hidden units (in the earlier example this could correspond to activations of 0.01, 0.03, 0.98, 0.02). These different ways in which to spread out the information over the hidden units affect the way features look: an example comparison between these two styles is illustrated in Figure 2.8.

A coding where only a few hidden units are active is referred to as a sparse coding, as opposed to a dense coding where many units are active at the same time. In Figure 2.8 you can see that the features learned in the sparse representation seem more 'elementary' to the problem at hand: you could almost hand-select a number of features to reconstruct a digit. The dense representation on the other hand, seems much more opaque in terms of understanding what is being encoded. In our music system this 'elementary' style of coding is much more desirable: we can get a more intuitive feel for what the system is encoding exactly. The fact that these features seem to represent separate building blocks can also improve the extent to which a network can reconstruct the input from the hidden unit activations (Goh et al. 2010).

The human brain also seems to use a (very) sparse coding, and the article we mentioned before, where Lee et al. (2007) compared features learned by an

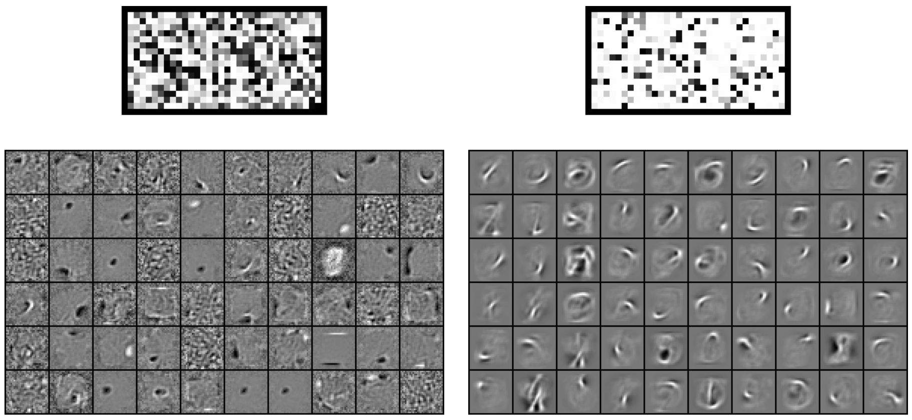

(**a**) Dense coding (**b**) Sparse coding

Figure 2.8 A comparison of features trained on MNIST using dense and sparse coding. The top rectangle visualizes hidden unit activations for a single sample (black for 1, white for 0) and the bottom rectangle visualizes a subset of features (light for positive weights, dark for negative weights). As you can see, with dense coding more hidden units are active at the same time, at more varying levels of activity.

RBM to characteristics of the V2 area of the visible cortex, indeed also used sparse coding RBMs. In nature sparse coding has the additional advantage that fewer neurons are active, and thereby less energy is being used.

One of the ways in which sparse coding is induced in the brain is through lateral inhibition: active neurons often suppress the activity of surrounding neurons. In a sense neurons compete for being active, where the ones that respond most specifically to the incoming signal win. In RBMs, implementing this in the same way would be rather difficult: we would need a complicated interaction between neurons in the same layer, while we can normally calculate their activations independently of one another.

Fortunately there are other means to accomplish the same. One particularly simple way to stimulate an RBM to learn a sparse representation is by penalizing it for its hidden unit activations: we add a term to the cost function representing the deviation from a target average activation (for example 0.05 for 5% hidden unit activation on average). This was done by Lee et al. (2007), and in some cases can work quite well. However, there is a downside: with

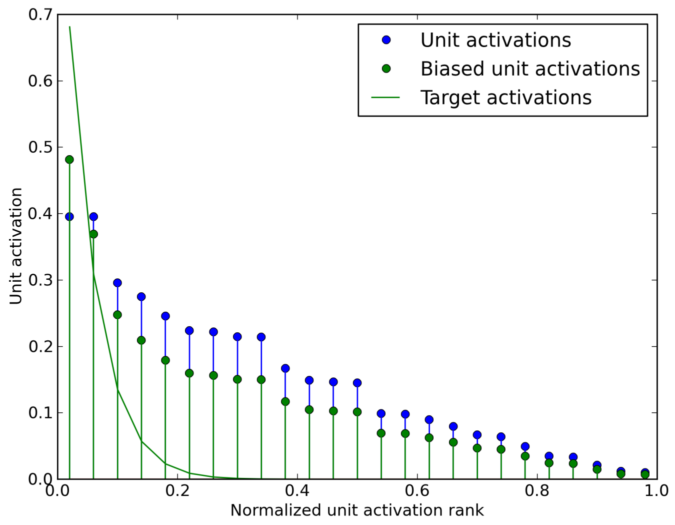


Figure 2.9 An illustration of the method for enforcing sparse features described by Goh et al. (2010). Hidden unit activations are biased to approach a target activation profile.

this implementation, you have no control over the exact distribution over the hidden unit activations – the cost function can be kept low either by setting 5% of the units to an activation of 1 and the rest to 0, or alternatively by setting all units to an activation of 0.05. The former is a good example of a sparse coding, the latter is a good example of a useless network.

Goh et al. (2010) proposed a solution to this, such that it is possible to control both the target sparsity, and how the activations are distributed among the hidden units. The downside is that it is computationally more expensive, making it slower in practice, but we can have some guarantee that the coding is sparse in the way that we like. Figure 2.9 illustrates the approach: first we sort our hidden unit activations for a single input sample to obtain a ranking of the unit activations. We then simply interpolate the true activations (blue dots) towards our desired activation pattern (the green graph). As you can see, the biased activations (green dots) look more like a sparse coding than the originals. By biasing the hidden unit activations in this way during training, due to the way the RBM learning algorithm works, the actual activations will tend to more and more closely resemble the target activations as learning progresses.

Beside a sparse coding, there is also such a thing as a selective coding. Where sparsity refers to the number of hidden units that are active for any

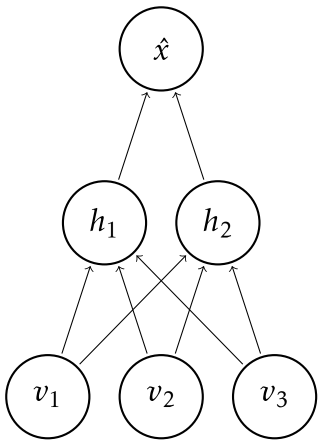


Figure 2.10 A simple FFNN with three input units, two hidden units, and an output unit.

given sample, selectivity refers to the fraction of samples for which a hidden unit is activated. Suppose we have a network in which one of the units is always on, and the rest are always off: in this case we have a sparse coding, but there is very little information in the coding because the output is always the same. In a sense, hidden units which are always on or always off are useless because they convey no information whatsoever.

Therefore, what we want is a coding where for any one sample only a limited number of hidden units are activated, and where for any one hidden unit it only activates for a limited number of samples. Luckily, both of these things can be achieved with the method developed by Goh et al. (2010), and this is the method we will use in the rest of this thesis.

2.2.8 Supervised learning with FFNNs

As was mentioned in Section 2.2.3, often unsupervised learning is used as an initial step before supervised learning. The techniques used for supervised and unsupervised learning are somewhat different, as are the considerations during training. In this section we cover some of the main issues to be taken into account when fitting the pre-trained network to target data.

When making the step to supervised learning, we add an output layer to the network, and train the network on data for which we know how we want the output activations to be. Figure 2.10 displays a very simple neural network consisting not only of input and hidden units, but also of an output unit, indicated with $\hat{x}$. The number of output units is equal to the number of parameters that the network should predict or estimate, in the case of MNIST the output layer would consist of 10 units, one for each digit 0 to 9, where the activation of the unit corresponds to the probability of the input being that digit.

If a network has been properly trained, one can set the input values of the network for a certain sample, and calculate the activations in the rest of the network. The output node activations then provide a prediction of the output value for which the network has been trained (for MNIST, a probability that the input data represents a digit from 0 to 9).

Depending on the kind of value that the user of the network wants to predict, different kinds of transfer functions can be used for the output layer. In this thesis, we are interested in real-valued performance parameters, which are not necessarily bounded by a certain range – in this situation a linear layer is a suitable choice, because it imposes no limit on the values that can be predicted (unlike a sigmoid layer, which outputs values between 0 and 1).

To obtain a functional FFNN, the system needs to be trained first, so that the weights actually allow the network to make a prediction based on the input data. For training, data is used where both the input values and output values are known (i.e. hand-written digits with their corresponding labels for MNIST). In the semi-supervised learning setting, all weight matrices except for the matrix determining the connections of the output layer are trained in an unsupervised setting. The top layer (the output layer) is then trained in a supervised way. The weights of the layer are adjusted to minimize the discrepancy between the predictions ($\hat{x}$) and true outputs (x) for the training data, where the discrepancy is commonly quantified by using the squared error $(\hat{x}-x)^2$.

Depending on the transfer and cost functions used for training, the regression of the top layer can be done in different ways. A common method that can be used with many different transfer and cost functions is gradient descent (often referred to as back-propagation). Given that we have a cost function C for our network (typically the squared error), we can calculate the gradient of the cost function with respect to each of the connection weights $\frac{\partial C}{\partial w_{ij}}$ and bias terms $\frac{\partial C}{\partial b_j}$. It is generally possible to obtain these gradients symbolically. Since $\frac{\partial C}{\partial w_{ij}}$ describes to what extent weight w_{ij} influences C, we can simply adjust w_{ij}

slightly to decrease the cost C:

$$w_{ij,new} = w_{ij,old} - \lambda \times \frac{\partial C}{\partial w_{ij}} \tag{2.4}$$

and likewise for the bias terms. Here, λ is the learning rate, which determines the step size we take during every iteration of the training procedure. As for RBMs, this is a hyper-parameter for which suitable values are generally found through trial and error, typically the value will be in the order 0.01-0.1. By iterating the weight updates many times, cycling through the training data, eventually a configuration of the network is found that (hopefully) captures the relation between input data and target values quite well. Gradient descent can also be used to fine-tune the weights in lower layers of the network that have been pre-trained in an unsupervised fashion.

In the case where the transfer function of the output layer is linear, and the cost function is the squared error, a faster approach is also possible by simply using least-squares to find a weight matrix for fitting the hidden activations to the output values. Whereas gradient descent requires many iterations and gradually 'homes in' on a solution, least-squares provides an optimal solution straight away.

2.2.9 Overfitting

When training a neural network, we make a certain assumption that patterns that seem valid and consistent in the training data, will be so as well in new data – this, we presume, allows the network to make valid predictions about new input for which we do not know the corresponding true output values. Although in many cases this assumption seems to work, it is quite dependent on factors such as the nature of the training data and the amount of training data.

A famous example where the nature of the training data caused a neural network to fail comes from the research in perceptrons (an early neural network variant). A claim was made that a system could identify whether photographs contained tanks (a quite advanced feature). Unfortunately, it turned out that the system had learned to distinguish between different kinds of lighting in the photographs, where in the training data all photos containing tanks were taken on a different day than those not containing tanks. Clearly, this is not pattern recognition in the way that we would like, as it no longer works on a different dataset. This implies that it is important that the training dataset is actually representative for real data, and if training data might be biased in some way, it is a good idea to test a network on data from a different origin.

The amount of training data can also influence the performance of a network. Particularly if a small amount of data is available, but the network contains many hidden units, the network can learn details about the training data that might not generalize to data outside the training set. Going back to the MNIST example, if in the training set there is only one sample for which pixel (17, 8) has the value 1, and this sample represents a 9, the network might learn a very strong correlation between this particular pixel value and the classification. Typically this would mean that a very strong weight would arise in the network, linking this pixel value to the classification as a 9, so that this one sample is always correctly classified. Again, this is not the kind of 'knowledge' that we want our network to encode, because the relation might not hold for new data.

One way to deal with this, which we briefly mentioned earlier, is to add a regularization term to the cost function. Typical terms are L2-regularization (imposing a penalty proportional to the sum of squared weights in the network) and L1-regularization (imposing a penalty proportional to the sum of absolute weights in the network). In this way, when a weight takes on a large value, the gradient is influenced by the regularization term in such a way that further growth of the weight parameter is counteracted. Variants of least squares, such as ridge and lasso regression (Tibshirani 1996), can be used for L1- and L2-regularization.

Another way to counteract overfitting is by stopping the training process before convergence. This does not apply to least-squares regularization (because this is not a gradual process), but is very common when using gradient descent. The typical approach when training a network is to split data for which input and output values are available into three subsets: a training set, a validation set and a test set. Only the training set is used for performing gradient descent, and during the training process, the performance of the network on the validation data is monitored. Initially, performance typically improves during training for both the training data and the validation data, until at some point performance on the training data improves but performance on the validation data deteriorates. At this point, the network is visibly overfitting (learning relations that hold in the training data, but that do not generalize to other data). Training is then halted. The reason for also having a test set is that both the training and the validation data have influenced the training procedure in such a way that performance on these datasets would be biased – therefore testing the true performance of the network (on general data) on these sets is not appropriate.

When using least-squares, the top layer automatically converges to the optimal configuration for the training data. There is no monitoring of the performance of the network using validation data, and only training and test datasets are necessary. On the one hand, this means the risk for overfitting is a bit higher. On the other hand, a least-squares fit is much more consistent than using gradient descent because we could repeat the fit and it will always result in the same solution (whereas gradient descent has an element of randomness). If there is enough training data, overfitting is not much of a problem, and there is still the possibility of adding a regularization term to the cost function.

2.3 Previous work

The following sections describe work that has been done before, analyzing the mechanisms behind musical expression (Section 2.3.1), and developing computational models that deal with music (Section 2.3.2).

2.3.1 Studies of musical expression

There are different kinds of accents musicians can use to add expression to their music (Parncutt 2003): time, pitch, loudness and timbre. Although for a large part, the musical score prescribes how these accents are to be placed, the musician still has significant freedom as well – and precisely this is what sets better musicians apart.

Although there do not seem to be clear-cut rules defining what is appropriate in musical expression, there are many patterns. One of the first researchers to formally study music performance was Seashore (1938), who measured tone durations of the same song sung by different artists. He recognized that although singers have their own style, there seem to be commonalities in how different singers perform a particular piece of music. Over time, many factors influencing expression in musical performances have been identified. Important factors are structure of music (i.e. musical grammar) and the emotion that a piece of music is intended to convey.

Many studies in musical expression are based on rigorous structural analysis of the music at hand, the idea being that just as in language, particular segments have a certain function, and that this affects the way that they are presented. A number of different theories for structural analysis of music have been developed over the years (Schenker 1969, Narmour 1977, Lerdahl & Jackendoff 1983, Lerdahl 2001). These theories mostly try to relate structure to the melodic expectancy of the listener, i.e. what a listener expects to happen after certain events (for example a certain step in pitch from one note to the next).

These structural models are hand-crafted, and are based on analysis of existing music. However, during music composition these models do not necessarily play an important role (they are somewhat after-the-fact). This makes it hard to state that these models precisely describe "what is going on in the music". Still, causal relationships between these methods of analysis and patterns of expression have been found by many researchers (Clarke 1988, Palmer 1989, Repp 1990, Sloboda 1983). Other research that reinforces the value of these structural analyses was published by Large, Palmer & Pollack (1995), who showed that there were significant similarities between substructures identified by structural analysis, and those stored in a neural network that contained reduced memory representations of the score.

Gabrielsson & Juslin (1996) studied the extent to which emotion affected music performance: they asked musicians to play the same piece with the intention of conveying different emotions, and asked listeners to identify these emotions (at which the listeners were relatively successful). Because the musical score did not change between experiments, it is clear that there is a lot of freedom in the performed interpretation of music, and that this affects the way it is perceived. Further research into the effect of emotion was done by Juslin & Sloboda (2001) and Juslin & Vastfjall (2008).

Although structure and emotion seem to be very defining in music performance, these are definitely not the only factors. For example, there seems to be a strong interrelation between different expressive parameters, particularly between tempo and timing (Repp 1994, Desain & Honing 1992, Desain & Honing 1994, Honing & de Haas 2008). This is not just a side-effect of the way music is performed, listeners really prefer the amount of expressive timing to change with tempo (Repp 1995). Many other factors influencing music performance are covered by Palmer (1997).

2.3.2 Computational models dealing with music

Different computational models of expression have been built relating recognizable structure in score to performance parameters. For example, Todd (1989) investigates patterns in expressive timing. Sundberg, Askenfelt & Frydén (1983) and Friberg et al. (1991) have developed systems to predict timing and dynamics using rules based on score properties. Bisesi & Parncutt (2010) also developed an accent-rendering system based on structural analysis of input score. Other rules were developed by De Poli, Irone & Vidolin (1990).

A drawback of all these expression systems is that they are based completely on hand-defined rules, which are in turn based on musicological analysis. However, it is hard to fully describe musical expression with hand-defined rules: many different rules would be required to capture the full range of possibilities, and it might not be realistic to formalize all possible expression in this way. These systems also inherently do not discover new rules for expression, apart from the rules which the designer of the system supplies. There have been a number of artificial intelligence (AI) approaches to overcome some of these limitations.

Widmer (1996, 2000, 2003b) used AI techniques to discover simple rules in musical performance, again based on basic structural analysis. Katayose & Inokuchi (1993), Arcos & De Mántaras (2001) and Suzuki, Tokunaga & Tanaka (1999) also take this approach. Bresin, De Poli & Ghetta (1995) take a fuzzy logic approach to relating performance to basic score properties, and Bresin (1998) relates basic score properties to expression using a neural network. Ishikawa (2000) takes an approach using multiple regression analysis.

Whereas the previously mentioned works base their predictions on an initial transformation of the input score to relevant parameters (structural elements or properties), Grachten & Krebs (2014) took one further step back. They applied feature learning to model expressive dynamics in piano music, comparing several different feature learning strategies: Principal Component Analysis (PCA), Non-negative Matrix Factorization (NMF), and Restricted Boltzmann Machines (RBMs). As input for unsupervised learning a 'note-centered' representation of the score is used, where features are later correlated to expressive parameters of the input notes using supervised learning. In essence, the system is programmed with no prior knowledge whatsoever, and only learns by example. This removes all dependency on existing musical theory. They concluded that in general, the RBMs performed best, and their set-up forms the basis for the experiments in this thesis.

Finally, work done by Boulanger-Lewandowski, Bengio & Vincent (2012) deserves mentioning: they developed a machine learning system for predicting harmonic progression in music. This system independently learns about structure in music using RBMs (it is not supplied with hand-designed features nor with a musicological analysis of the input), however it is not used for performance analysis. Still the underlying mechanism is similar to what we aim to accomplish. Other relevant previous work is that done by Grachten & Widmer (2011), who developed a model to predict expressive parameters from score annotations. This system is reasonably successful, but inherently misses out on opportunities to learn about immanent expression (expression that is not explicitly defined in the score).

Whereas most of the mentioned studies focused mostly on qualitative models of expression, relating events in a musical score to events in a performance, the approaches by Grachten & Widmer (2011) and Grachten & Krebs (2014) were more quantitative in nature – they could actually predict expressive performance parameters in data describing true performances. We aim to do the same in this thesis, and in fact we also make use of the same data.

Chapter 3

Architecture and workflow

This chapter describes the overall set-up of the machine learning system. Two models were developed, one where the score is considered one note at a time, this is described in Section 3.1, the other considering the score one beat at a time, described in Section 3.2. In Section 3.3 we cover the technical implementation of the model, and Section 3.4 explains how the performance of the model is finally evaluated.

The reason two different models were developed is that different expressive parameters have different characteristics: dynamics/loudness is most simply described as note velocity (a measure for how 'fast', or vigorously, a note was struck). This means we have one target value per note. However, when we look at tempo, considering tempo as a single value per note is somewhat awkward: tempo in a piece does not really change from note to note (also consider the situation where multiple notes sound simultaneously), and it is not so much related to specific note events – it is more something that changes over the course of time. For that reason it makes more sense to relate tempo to the structure of a beat, a measure or a longer stretch of music. Timing on the other hand is again more suited to a representation where a value is determined for every note.

3.1 Note-by-note model

In this section we describe the details of the model that considers the data one note at a time. We start by describing the way in which the model 'sees' the music data (Section 3.1.1), followed by a description on how we use machine learning to find structure in this data (Section 3.1.2). Finally we compare the characteristics of the model developed by Grachten & Krebs (2014) to ours (Section 3.1.3).

3.1.1 Input data

The available music and score data cannot be used for machine learning directly – we need to transform the data to be able to learn from it. In this section we describe how the input samples are represented (which will result in visible unit activations in the neural networks), and how we determine the corresponding target values (the true values which we want the system to predict in the output units). Note that this section is not about the particular musical data we use as input – the choice of datasets for training is described with the experiments, and the characteristics of the datasets are described in more detail in Appendix A.

Note-centered representation For predicting velocity the input features for the system were designed mostly in the same way as done by Grachten & Krebs (2014), i.e. applying a note-centered representation. For every note in the musical score, a sample is generated representing a grid of the note and its surrounding notes, providing a representation of the context of each note. This is visualized in Figure 3.1. Just as was described in Section 2.2 for training on MNIST, the model sees the input samples as sequences of 1's and 0's, the spatial structure we see in the figure is in no way apparent to the machine learning system.

Each sample is effectively a slice from a piano roll of the music – a grid where the horizontal dimension represents time and the vertical dimension pitch. Each row in the grid is a semi-tone apart from the rows below and above in terms of pitch, and each column corresponds to a time-span of 1/8th beat (typically the duration of a note with length 1/32, depending on which note duration corresponds to a beat, which is indicated in the raw score/MIDI data). Note that there is overlap between the samples describing different notes.

The reason a note-centered approach is applied is that it is insensitive to shifts in absolute pitch: if a piece is pitch-transposed by 1 semi-tone, the samples still center on the notes, thereby shifting the slices by a semi-tone as well. Also, if, for example, a major triad occurs in the score in different locations with different root notes, the notes in the different triads will be represented similarly in the note-centered representation, meaning that the model can learn about a sort of generalized notion of major triads rather than learning about A-major triads and D-major triads separately. This rather elegant solution implements in the model a characteristic that holds for people as well: being sensitive to relative changes in pitch and timing rather than abso-

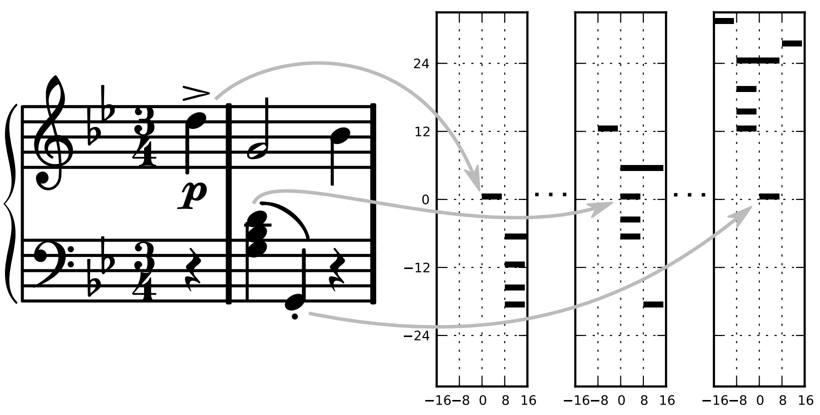


Figure 3.1 The note-centered representation. For each note in the score a sample is generated where this note is in the center and surrounding notes in terms of pitch and time are visualized in their relative positions. Black values correspond to 1 and white to 0, the horizontal axis corresponds to time and the vertical to relative pitch. Figure by Grachten & Krebs (2014), used with permission.

lute changes (just as most people do not have absolute hearing, but are quite sensitive to relative pitch intervals).

In our model, notes occurring up to 3 beats before or after the onset time of the centered note are included in the grid (i.e. the grid is $6\times8 = 48$ pixels wide) and notes within 55 semi-tones in terms of pitch (the grid is $2\times55 = 110$ pixels high). This results in $6 \times 8 \times 2 \times 55 = 5280$ different visible units for neural networks trained on this representation. Note that this is different from the dimensioning illustrated in Figure 3.2. A random selection of actual samples from Chopin's music can be seen in Figure 3.2.

Note length is encoded in the note-centered representation: longer notes take up more pixels horizontally corresponding to their duration. One remaining detail to mention, is that the last pixel for each note is set to 0: this is done so that the model can distinguish between situations where there is a long note sounding, and where there are consecutive short notes of the same pitch. If this was not done, the model would for example not be able to see the difference between two consecutive eighth notes of the same pitch and a single quarter-note.

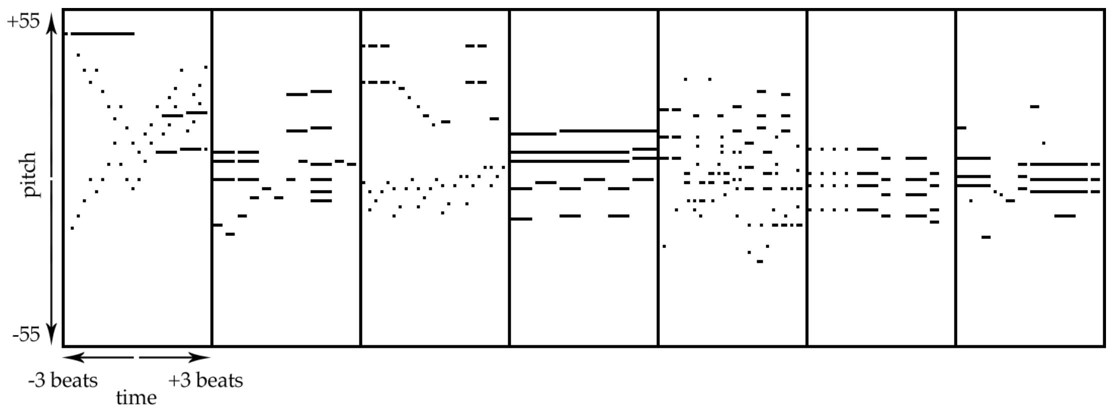
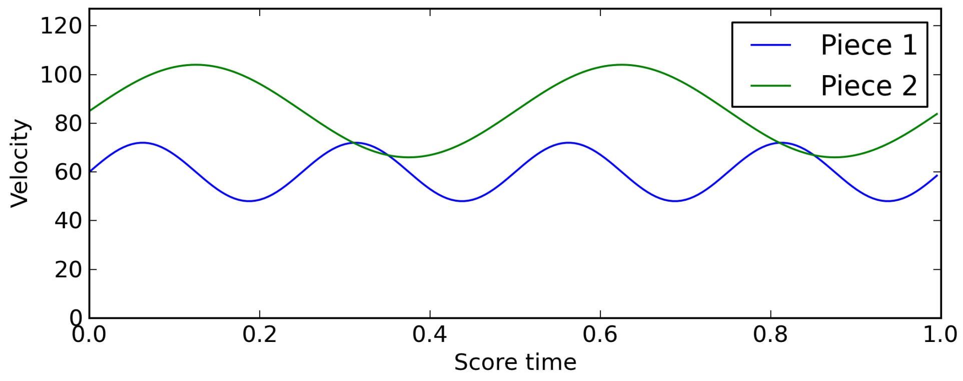


Figure 3.2 Visualization of a random selection of samples from the Magaloff (Chopin) data.

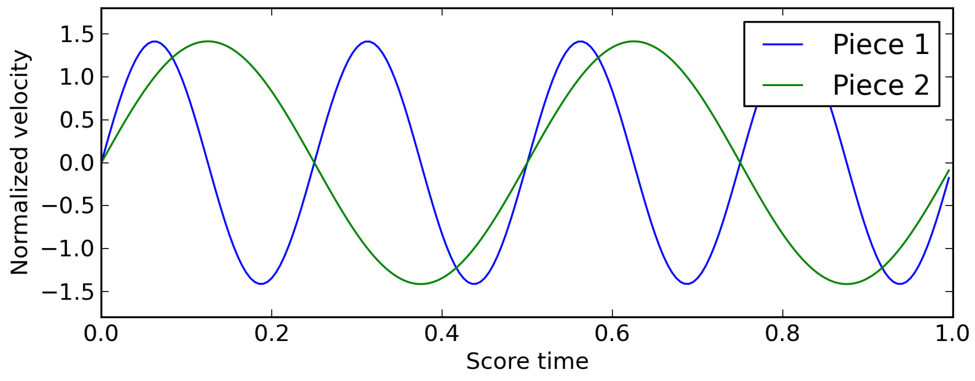
Velocity target values Now that we have described part of the data that will determine the input for the model, it is time to consider the output. One of the expressive parameters for which we use the note-by-note model is note loudness. In the performance data, values concerning loudness are stored as velocity values between 0 and 127 (where notes with higher velocity values were struck harder). However, it is inconvenient to fit directly to velocity: just as with our input pitch and timing, we are interested in relative velocity more than in absolute velocity.

We illustrate the concept behind the velocity normalization in Figure 3.3. In Figure 3.3a we visualize the velocity from begin to end of two hypothetical pieces. The visualization is purely for instructive purposes, in reality velocity variations in a piece are much more erratic. We see two different pieces, where each has a different average loudness, and also a different amplitude of variation (piece 1 has less dynamic range than piece 2). When considering which part of this signal is interesting for us to model, it is much more the variation of the velocity within the piece over time than the characteristics of the piece as a whole – the dynamic range and average loudness can be seen as characteristics of the piece as a whole rather than as local parameters. This is somewhat of a simplification, but we will see later that this in fact improves the results obtained with our model, suggesting that the assumption is not unreasonable.

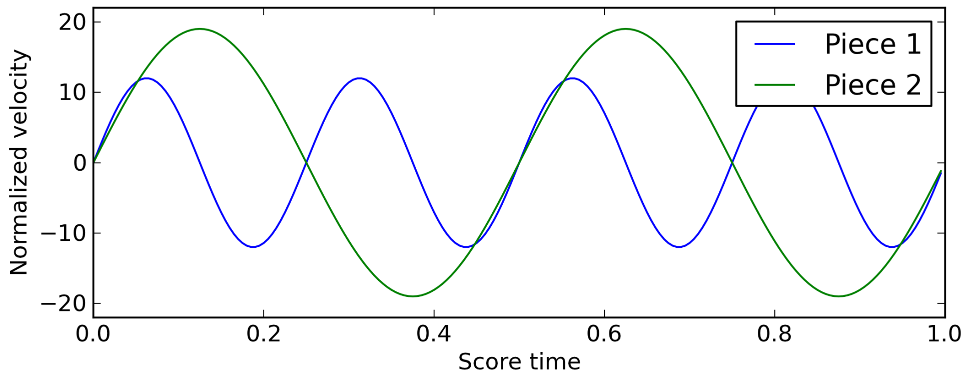
Because we want to study the trends in local variations within a piece, it makes sense to normalize the data in such a way that the model sees the local variations separate from properties of a piece as a whole. Our first normalization method, which we will refer to as N1 is based on this concept. The target



(a) Hypothetical performance data.



(b) Hypothetical performance data after N1.



(c) Hypothetical performance data after N3.

Figure 3.3 A visualization of normalizations N1 and N3.

values are determined as follows:

$$v_{N1} = \frac{v - \bar{v}_{piece}}{\sigma_{v,piece}} \tag{3.1}$$

where v is the note velocity, $\bar{v}_{piece}$ is the average velocity in the piece and $\sigma_{v,piece}$ is the standard deviation of the velocity values of the piece. Figure 3.3b illustrates what this transformation does to our hypothetical data: both mean and dynamic range are normalized away, and the signal represents the relative velocity variations within a piece. Given a model trained on data normalized with N1, if we predict velocity values for a new piece of score, we obviously predict values in the same 'style', so there is no prediction on what would be the average velocity, or the dynamic range of a piece. When rendering a performance in this way, a user would choose an effective $\bar{v}_{piece}$ and $\sigma_{v,piece}$ to recreate a full performance, in essence these are parameters that a user can tune to add more or less expressive variation to a piece, or to make it louder or softer overall.

One could argue that the dynamic range of a piece is also a musically relevant parameter – it would be nice if the model could provide an estimate of whether loudness variations in a piece are expected to be strong or moderate. We attempt to do this with N2, which is not visualized. A model trained for N2 is in fact also trained on the target values v_{N1}, but an additional target value is supplied: for every note $\sigma_{v,piece}$ is supplied as a target value, corresponding to the piece it is in. When predicting $\hat{\sigma}_{v,piece}$, our model now generates many predictions: one for every note. However, it is a piece-wide parameter, so we need only one estimate – the simple solution is to just take the average prediction from every note to determine the estimate for a piece as a whole. Given this dynamic range parameter, we can then provide a slightly more sophisticated prediction of the velocity of the note, where a user would only need to supply a mean offset.

Both of the normalization methods described so far are different from the method used by Grachten & Krebs (2014). To be able to compare results, we also use their normalization method, which we refer to as N3:

$$v_{N3} = v - \bar{v}_{piece} \tag{3.2}$$

which is visualized in Figure 3.3c. As you can see, differences in average loudness between pieces are normalized away, but differences in dynamic range are still apparent in the performance data.

When using N2 and N3 the prediction generated is actually more or less the same: an estimate of the velocity where only mean velocity for a piece is normalized away. However, in N2 the dynamic range (standard deviation) is predicted separately, and a mean is taken over all estimates. In N3 an estimate for the dynamic range is implicitly encoded, by the fact that it is still present in the target data.

The hypothesis is that N1 is easiest to learn: some sources of variation are normalized away, which might make it easier for the model to understand what is going on in the data. Also, N2 is expected to perform better than N3: if many data points are used to estimate the dynamic range in the piece, and these estimates are then averaged, this is likely to be a better estimate than effectively estimating the dynamic range for every note separately.

Velocity-history representation In the note-centered representation, a lot of information is encoded about the harmonic and rhythmic context of a note. However, something else that is interesting to consider is the expressive context of a note: the expression of a single note cannot be seen as a completely independent event, in fact there is good reason to expect a quite strong relation between expression of a note and the expression of the notes before it. A simple example is a crescendo, where consecutive notes are played louder and louder: if we know the previous notes were increasing in loudness, this might be a signal that the note we try to predict next will be even louder. Many more patterns like this can be imagined, for example, a strong relation between the loudness of a note exactly one measure or one beat before the current note.

To encode this sort of information, an additional representation of the performance data is used. In this representation, samples are made up of a grid representing the loudness with which previous notes were played, and we refer to it as the velocity-history representation. A few random samples from the Magaloff dataset are displayed in Figure 3.4.

The velocity-history and note-centered representations are conceptually similar, but instead of displaying note pitch on the vertical axis, the velocity-history representation displays normalized velocity. It does not contain notes from the future: it only contains the notes that have been played in the past 3 beats. The horizontal axis relates to the time component in the same way as in the note-centered representation, i.e. every column corresponds to a fixed time interval in score-time (1/8th beat).

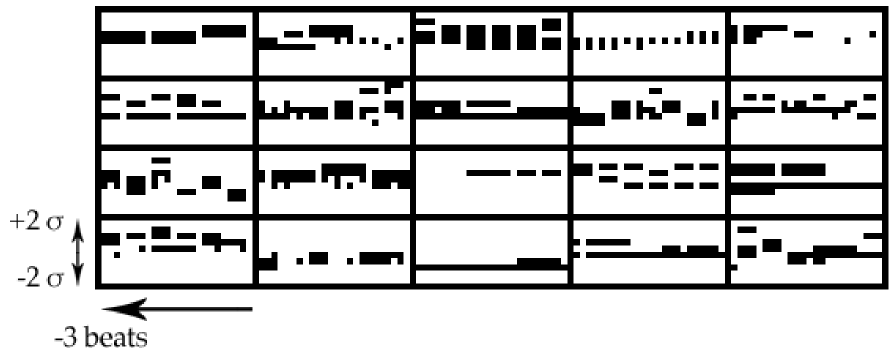


Figure 3.4 Samples in the velocity-history representation. The total number of input units when using this representation is $12 \times 3 \times 8 = 288$.

Along the vertical axis the normalized velocity of past notes is depicted, using N1. A normalized velocity of 0 thus corresponds to the average velocity, a value of 1 to 1 σ above the average velocity, etc. In the velocity-history grid there are 12 rows, corresponding to the range -2 to 2 in terms of the normalized values, which correspond to the velocity values between $2\sigma_{v,piece}$ above and below the mean velocity of the piece. Notes for which the velocity falls outside of this range are clipped to the highest or lowest value, depending on where they fall outside of the range.

Something very important to note about this representation, is that to generate it you in fact need performance information for previous notes. This can be done in two ways: one can take true performance information from available performances, or one can take performance information predicted by the model previously. In the first case, if we only want to explain why a note in a performance is played in a certain way, this is acceptable. But if we want to predict an entire performance, we can of course not assume that true performance data is available, because for new data it is not – in this case we have to use model predictions to generate velocity-history samples for subsequent notes. This is described in a bit more detail in Section 3.1.2.

Timing target values We also use the note-by-note model for predicting timing of individual notes. Tempo and timing are very fuzzy definitions, and it is not clear where one begins and the other ends – one simply needs to choose a definition for the parameters and stick with this. In this thesis, we determine tempo from the time progression of a piece in a smoothed way (explained in more detail in Section 3.2.1). Essentially, for each onset in score time (which is measured in beats), we have an expected onset in performance time (which is measured in true time, i.e. milliseconds), based on the observed tempo 'frame' of the piece.

Still, because this frame is smoothed, individual onsets in performance time do not occur exactly at the expected time. This discrepancy is what we define as timing:

$$\Delta t = t_p - \hat{t}_p \tag{3.3}$$

where t_p is the observed onset time (in performance time, i.e. milliseconds) and $\hat{t}_p$ is the expected onset time (again in performance time). Δt is then a value for each note which relates to its timing relative to the tempo frame, and this is what we use as target value during training.

As is described later in Section 3.2.1, there are some problems with the tempo and timing information in the performance data. For the timing data we deal with this by using outlier detection – extreme timing values (further than 3 σ from the mean) are taken out in three iterations.

3.1.2 Machine learning set-up

Now that we have determined what kind of input samples we want to supply to the model, and what kind of target values (output data) to fit it to, it is time to describe the neural networks that will encode the relation between the input and output data. We apply semi-supervised learning, meaning that we first perform unsupervised learning on the input samples only, followed by supervised learning with both input samples and target values. Figure 3.5 illustrates the final structure of our neural network, the following sections describe how we train the different layers.

Unsupervised phase During unsupervised learning the objective is to find recurring patterns in the data, and store this in the structure of our neural network. We want to find features that can be used to compactly describe our input data (the musical data in note-centered and velocity-history representations). This is the same principle as was illustrated in Section 2.2 for character recognition, but now we use musical score data.

For the unsupervised learning, sparse RBMs are used, with the sparsity mechanism proposed by Goh et al. (2010). The note-centered and velocity-history representations of the input data are considered separately, i.e. separate networks are trained on the musical data in the two representations. Only

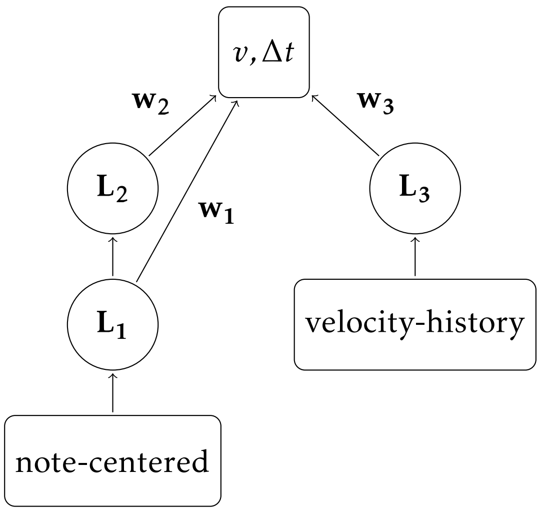


Figure 3.5 Diagram of the architecture used for velocity and timing prediction. Hidden unit activations are sigmoidal, the top layer is linear (and fit using least-squares).

during supervised learning the feature activations of these networks will be combined for making our velocity and timing predictions.

In Figure 3.5 three layers correspond to hidden units trained in an unsupervised manner: $\mathbf{L_1}$, $\mathbf{L_2}$ and $\mathbf{L_3}$. $\mathbf{L_1}$ is a layer of 512 hidden units that is trained directly on the input data in the note-centered representation. $\mathbf{L_2}$ is a layer of 200 hidden units trained on the feature activations in $\mathbf{L_1}$, making the combination of $\mathbf{L_1}$ and $\mathbf{L_2}$ a deep network as described in Section 2.2.3. $\mathbf{L_3}$ is a layer of 120 hidden units that is trained on the input data in the velocity-history representation.

The reason these dimensions were chosen is that for L_1 in the experiments of Grachten & Krebs (2014) around 500 features presented a nice trade-off between descriptive capacity and limited requirement of computational resources, the same holds for around 200 features in L_2. For the features trained on the velocity-history representation the number of units was chosen with the intention of ensuring an information bottleneck, while keeping enough hidden units to still have some descriptive capability. However, as always choice of hyper-parameters is somewhat arbitrary – it might be possible to improve the performance of the model through fine-tuning of these parameters.

Supervised phase After learning features, we have a system that transforms input score into a different representation. Hopefully, the alternative representation contains meaningful information about the input data, and with RBMs often it does, but there are no guarantees. (Spoiler: in our experiments the representation does seem to be meaningful.) The next step is to relate this alternative representation (the feature activations) to target values.

Many of the experiments in this thesis relate to different ways we set up the supervised phase: using different datasets for training, and using different sets of feature activations for learning. However, a number of things are always the same: we always use a linear layer for the output, and we always fit using least-squares (see Section 2.2.8).

As target data we use the velocity (v_{N1}, $\sigma_{v,piece}$ and v_{N3}) values described earlier and the timing values (Δt). Clearly we can only perform supervised learning with data for which we have both the score and the corresponding performance parameters – the only datasets where this is the case are the Magaloff and Batik datasets. However, as mentioned before, we use different subsets of the data for different experiments.

During training, our least squares fit determines the best sets of values for the weight matrices $\mathbf{w_1}$, $\mathbf{w_2}$ and $\mathbf{w_3}$ for our training data, which then also enables us to predict output values for new input score. However, again, we sometimes use subsets of the full architecture. We will refer to different configurations by the layers for which the activations were included in the feature matrix during training: $\mathbf{L_2}$ refers to an architecture where the target values are fit only on the hidden unit activations in layer $\mathbf{L_2}$, whereas $\mathbf{L_1L_2L_3}$ refers to the full architecture. Effectively, if we use only $\mathbf{L_2}$, we try to find the optimal weights for $\mathbf{w_1}$, $\mathbf{w_2}$ and $\mathbf{w_3}$ given that $\mathbf{w_1}$ and $\mathbf{w_3}$ are zero matrices.

Feedback model In Section 3.1.1, we mentioned that there are different ways for generating the velocity-history data: using velocity values from true performances, and using velocity values predicted by the model itself. These two approaches effectively correspond to the two different objectives we stated for this thesis: rendering performances, and explaining existing performances.

When rendering performances, we have only bare score available, for which we want to give an estimate of how the expressive parameters change during the course of the piece. We know nothing about real (human) performances of the piece, which means that true velocity values are simply not available for generating the velocity-history representation. In this case, we can generate velocity predictions for the piece one note at a time, and use these predictions to generate the velocity-history representation for consecutive notes, i.e. feed back predictions into the model. We will refer to this model as the **VHF** model – it uses all hidden layers ($\mathbf{L_1L_2L_3}$), with the velocity-history input for $\mathbf{L_3}$ generated from previous predictions.

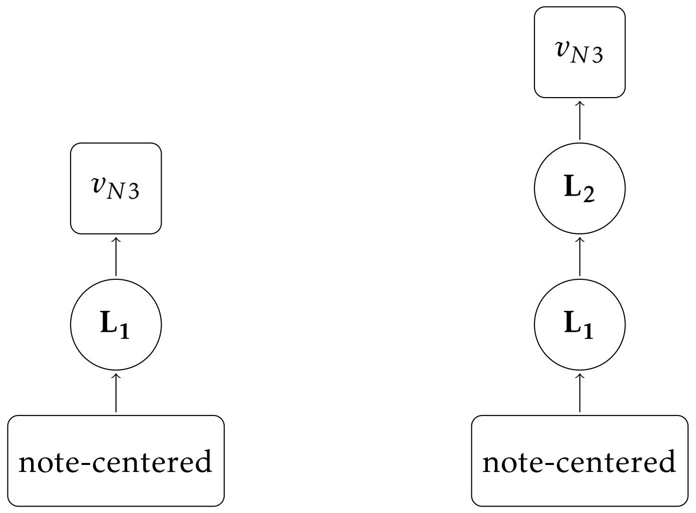
Considering that the model has been trained on data from real performances, it might recognize expressive patterns the previously predicted note velocities, and this might add some consistency to the predictions: new velocity predictions take into account what the model predicted before. On the other hand, if the model makes a wrong prediction early on, this can affect later predictions, which can in turn affect later predictions resulting in a sort of snowball effect where errors propagate all the way through the score. This limits the use of the velocity-history representation for rendering in terms of how well it can do.

When considering analysis of a performance (explaining its expressive variation), we do have a rendered performance available: this means we can use velocity values from the original performance to generate the velocity-history samples. This makes the model a sort of phrase-completion system: given that the past few notes were played at certain velocities, it should predict the velocity of the next note, after which we can compare the prediction to the ground truth and see how much of the variation in the ground truth we can explain. In this case the representation contains a lot of information about the piece: it already conveys to the model to a large extent whether it is in a loud or soft part of a piece, whether it is in a crescendo (increasing loudness) or decrescendo (decreasing loudness), etc. We will see later that in fact, this model can explain a large part of the variation in performances. However, it is very important to realize that these are predictions of single notes considering their expressive context as known, not predictions of entire pieces.

3.1.3 Comparison with Grachten & Krebs (2014)

As was stated before, the experiments described in this thesis are very similar to the ones performed by Grachten & Krebs (2014). However, there are a number of differences, some related to the way the experiments were done, and some to the way the neural networks were constructed.

In their implementation of the note-centered representation, the pitch range always accommodated the full range of possible relative pitch values: in case



(**a**) Single-layer RBM architecture. (**b**) Stacked RBM architecture.

Figure 3.6 Diagrams illustrating how the architectures used by Grachten & Krebs (2014) compare to ours.

of a note with the lowest possible pitch, their representation would still be high enough that the highest possible pitch is still visible in the representation. We limited this range to 55 semi-tones above and below the centered note, resulting in smaller features, less input units and faster computation. The situations where this range cannot accommodate all sounding notes are rare, and are also unlikely to influence the learned features.

In terms of architecture, Figure 3.6 illustrates the architecture used by Grachten & Krebs (2014) (using our notation). They predicted fewer different target values (only the velocity values corresponding to our N3 normalization), did not use the velocity-history representation, and when using $\mathbf{L_2}$ never combined these hidden unit activations with the activations of $\mathbf{L_1}$. They also did not use sparse RBMs. Finally, for unsupervised pre-training of their network they used a much more limited dataset – only the Magaloff data where we use data from a number of different sources additionally.

3.2 Beat-by-beat model

The second model is built not to relate expressive information to particular notes, but rather to particular positions on the score timeline. This relates more naturally to tempo data than the note-by-note approach. The following sections describe how we structured this alternative approach: in Section 3.2.1 we describe how the performance and score data are transformed for training, and Section 3.2.2 describes the structure of the machine learning system.

The choice for a 1-beat segmentation was based on the fact that the input data is consistently segmented into beats (i.e. we know exactly where the boundaries of all beats are). If the data was known to be consistently segmented into (for example) measures of 3 beats, this would have been a possibility as well, but the data is strongly heterogeneous in that sense – one cannot expect to put data of 4-beat measures into an architecture designed for 3-beat segmentation, and still obtain consistent features. Finally, while timing takes place at a rather fine timescale in our definition, because of the fact that we see tempo as a variable that changes smoothly rather than abruptly (which is described in more detail in the next section), it seems unnecessary to segment the score into finer fragments.

3.2.1 Input data

We only use the beat-by-beat model for tempo estimation. We use only one input representation, again based on the piano roll of a piece of score but not centered on separate notes. The following sections describe how we define our input samples, how we determine the tempo information from the performance data, and how we deal with problematic data in the available performances.

Score representation The input representation in this model is rather simple, as it very closely resembles the input score as a piano roll. At the lowest level, score is input in score fragments spanning one unit of time, as defined in the input score (e.g. MIDI). The beginning and end of the fragment both fall exactly on a beat, and in Figure 3.7 the horizontal axis corresponds to time. The grey box indicates a fragment of score of one beat duration. The vertical axis corresponds to pitch, and unlike in the note-centered representation, the pitch is absolute: for every absolute pitch, there is one row in the representation matrix (although just as for all other representations we encountered so far, the feature in the grey box would again be presented to the model as a sequence of values without apparent structure rather than as a grid).

The fact that this model is sensitive to absolute pitch is not a 'feature' – it is a limitation. Initially experiments were done with different architectures with the goal of making the system insensitive to absolute pitch, the basis for

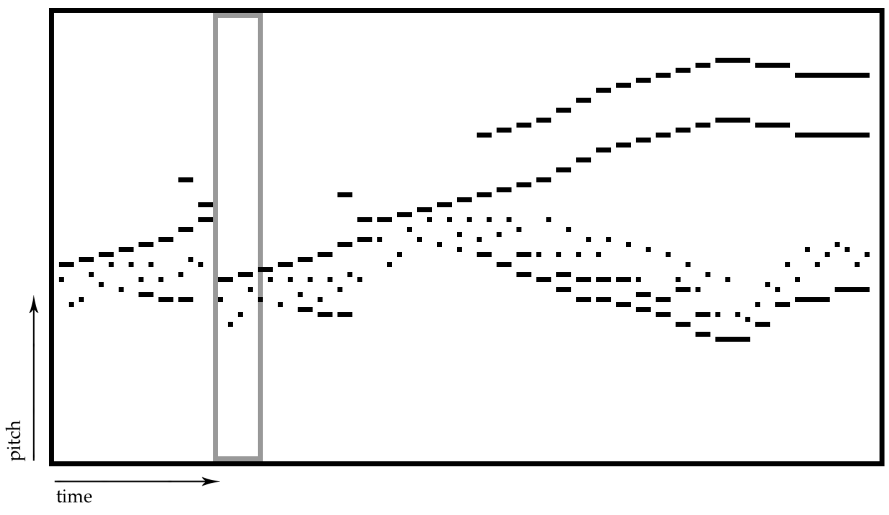


Figure 3.7 A piece of score (part of Chopin's Opus 1) converted to a binary grid, and a 1-beat segment (inside the grey rectangle). A 1-beat segment has dimensions 8×128 resulting in 1024 visible units.

this architecture is described in more detail in Appendix B, but despite some promising experiments the architecture was not mature enough for integration into the final system.

At a higher level in the model, multiple of these input samples are aggregated in the order in which they are found in the score (just as the architecture described in Section 2.2.6). At the higher level, 12 consecutive score segments of 1 beat length are combined. The choice for a 12-gram was based on the fact that it is often uncertain whether input data is in duple or triple meter, and a 12-gram will generally contain an integer number of measures in either case. 12 beats is also roughly the timescale for which we are interested in relating features to tempo changes.

Tempo performance data We derive our 'true' tempo estimates from performance datasets, describing music by Chopin (performed by Nikita Magaloff) and Mozart (performed by Roland Batik). These datasets link musical score notes to actual recorded performance notes – for any note in a piece, we can determine its position in the score (i.e. on- and offset time in beats), and the actual way it was rendered (on- and offset time in milliseconds, and velocity). The technical characteristics of the two datasets are very similar, and the Magaloff dataset is described in more detail by Flossmann et al. (2010). The musical characteristics of the datasets are described in more detail in Appendix A.

Although the datasets are among the largest and most detailed of their kind (if not the largest and most detailed), the datasets are not perfect. A large number of notes were matched automatically between score and performance, and also the recorded performances were not perfect, so some problems were unavoidable. For note velocity (dynamics) these errors do not lead to significant problems, but when using the data for tempo and timing estimation some issues arise. Some problems in the dataset originate in performance errors, while other problems are caused by a mismatch between the performance and the available score, or by technical issues. Four main problems can occur in the data for a particular piece:

- Extra repetitions: pieces can be repeated more often in the performance than in the score. In this case some performed notes cannot be matched to score notes, and a gap occurs in the performance onsets, possibly resulting in very low values for tempo.
- Skipped score: short pieces of score can be skipped, or repeated fewer times in the performance than in the score. In this case performance time progresses smoothly during a jump in score time, possibly resulting in very high values for tempo.
- Performance errors: a note can be played before another note, while in the score the ordering is the other way around. If left untreated, this can result in negative tempo values which are highly undesirable.
- Grace notes: these are considered infinitely short in the score, but do actually take up a finite amount of time in the performance. This can also cause some problematic tempo estimates.

The problems with extra repetitions are difficult to single out (but are rare) and remain in the data. When score is skipped, this is visible as a gap in the score when the corresponding features are generated: when a gap in tempo estimates arises, the estimates which pose a risk are flagged and later removed. Negative tempo values are also easily isolated and removed, this is described in the next section. The main problem with grace notes arises at the end of pieces, when there is a stretch of notes annotated as grace notes finishing off the piece – in these situations the tempo for the last beat is extrapolated from the rest of the performance. Unfortunately problems arise again during performance rendering (due to the infinitely short duration), these issues are unavoidable.

Tempo target values To be able to train our model, we need to extract target values representative for the tempo in a particular beat, which we can feed into the model in combination with the input representation. However, as described previously, the available data does not describe tempo: it describes on- and offset time for notes. This means a conversion is necessary. We will now describe how this performance data consisting of performance onset times (which we will refer to as t_p) and score onset times (which we will refer to as t_s) are converted into tempo target values. Only note onsets are taken into consideration, offsets are ignored (for piano music these in fact also have a smaller impact, as the volume of a struck note fades away over time).

As was stated before, it is hard to define the distinction between tempo and timing. In Section 3.1.1, we defined timing in terms of the difference between the observed note onset and the expected note onset, where the expected note onset was determined from a smoothed tempo frame. This tempo frame relates the progression in score time (in terms of beats) to the progression in performance time (in terms of true time, i.e. milliseconds), and we dedicate the next few paragraphs to the method for determining this tempo frame.

Due to the nature of the data, many notes can have the same corresponding score time t_s: for example for a chord at 8 beats in the score time (i.e. 8 quarter notes into the piece), all notes in the chord have the same t_s. Clearly this is a common occurrence. On the other hand, even though the performance time t_p is also a discrete (digital) value, detail is very fine, with a resolution of milliseconds and less. Notes played during a real performance typically do not occur at exactly the same measured t_p, even if they sound together as a chord.

To determine our smoothed tempo frame, we associate an expected performance onset time to every score onset time, i.e. we match each beat to a performance time in milliseconds. This tempo frame is determined in multiple steps, where each consecutive step uses the updated t_s and t_p values resulting from the previous step:

1. For each t_s at which multiple notes occur simultaneously, the average t_p of all corresponding notes is set as the expected performance onset time. Because we are now no longer considering true t_p for specific notes, but an estimated t_p for the point in score time, from now on we will refer to the estimated t_p as $\hat{t}_p$.

2. For each t_s where the corresponding $\hat{t}_p$ is smaller than that of the preceding t_s, the data point is removed (this can occur when due to a performance error a note is accidentally played before another note that is earlier in the score).

3. Thus far, we only have $\hat{t}_p$ values for the t_s values where there actually is a note in the performance. However, not every possible score time corresponds to a sounding note. The score time grid is now refined, such that for each possible t_s value in the duration of the score (with a resolution of 1/8th beat), a corresponding $\hat{t}_p$ is calculated by interpolating linearly from the surrounding data points. This means, for example, that we now also predict the progression of performance time during rests.

4. The beginning and the end of the grid are extended by 5 beats of padding, for which the $\hat{t}_p$ values are extrapolated from the first and last observed $\hat{t}_p$ values in the performance data (this corresponds to the average tempo of the piece).

5. We now have a homogeneous tempo frame: for a series of t_s encompassing the entire piece, we have a single $\hat{t}_p$ value for each t_s, at consistent intervals of t_s (1/8th beat). Because the frame is now in a consistent format, it is possible to smooth the $\hat{t}_p$ values: this is done using a moving average – a two beat long uniform smoothing window (a 17-element vector $\left[\begin{array}{ccccc} \frac{1}{17} & \frac{1}{17} & \cdots & \frac{1}{17} & \frac{1}{17} \end{array}\right]$).

We now have a $t_s \rightarrow \hat{t}_p$ mapping that corresponds to what we will consider as the tempo frame of the score, where $\hat{t}_p$ is the expected performance onset time (while t_p for a note is its true, observed, performance onset time). The $\hat{t}_p$ values resulting from the last step are the ones that we use for determining the timing values for individual notes in Section 3.1.1.

Still, at this point we only have a score time to performance time mapping – the values still do not correspond to tempo (which is the rate at which score time progresses compared to performance time). Obtaining the targets used to train the tempo model requires a few more steps:

1. Using the tempo frame, a value for tempo can be obtained for a short part of the performance: one only needs to divide the difference in t_s at the start and end of this short piece by the difference in $\hat{t}_p$ at the start and end. This is done to obtain a tempo value for each beat.

2. These tempo values are normalized to a mean of 1 for the whole piece. A value of 1 at a certain beat then corresponds to the average tempo of the piece, and other values correspond to the relative tempo (i.e. a value of 0.5 corresponds to half of the average tempo, and a value of 2 to double the average tempo).
3. In order to ensure that the model considers a transition to half of the tempo or to double the tempo as roughly equivalent (intuitively they are, but numerically one is 0.5 away from the reference and the other is 1.0 away from the reference) we take the base 2 logarithm of the given value.

The result is a value for each beat, which is representative for the tempo at that moment in the piece. This is what is used as the target during learning.

3.2.2 Machine learning set-up

We covered the input representation of the score and the determination of the target values. Now it is time to describe the machine learning architecture used to relate structure in the input to the target values. We will describe the unsupervised and supervised parts of the learning procedure separately, the overall structure of the network is visualized in Figure 3.8.

Unsupervised phase As is visible in Figure 3.8, an N-gram setup is used based on the same concept as that described in Section 2.2.6. The $\mathbf{L_1}$ weights are determined by training on input samples describing a score segment with a 1-beat duration (1024 input units). These weights are then replicated, so that in the larger network hidden unit activations through the $\mathbf{L_1}$ transformation are determined for 12 consecutive input samples (12 consecutive beats). Each of the sets of $\mathbf{L_1}$ activations has 240 hidden units. The $\mathbf{L_2}$ layer was trained on the hidden unit activations of the different $\mathbf{L_1}$ instances on a 12-beat sequence of input. $\mathbf{L_2}$ consists of 512 hidden units (for $12 \times 1024 = 12288$ input units).

Just as was done in the dynamics experiments, sparse RBMs were used for training with the sparsity mechanism developed by Goh et al. (2010). Also, again the dimensioning of layers (numbers of hidden units) was a trade-off between implementing an information bottleneck and having enough units for descriptive transformations – in this case just as in the note-by-note model it might be possible to improve the model by fine-tuning these parameters.

Supervised phase During supervised training we fit the tempo target values to the hidden unit activations in $\mathbf{L_2}$. We only use the full set-up, we do

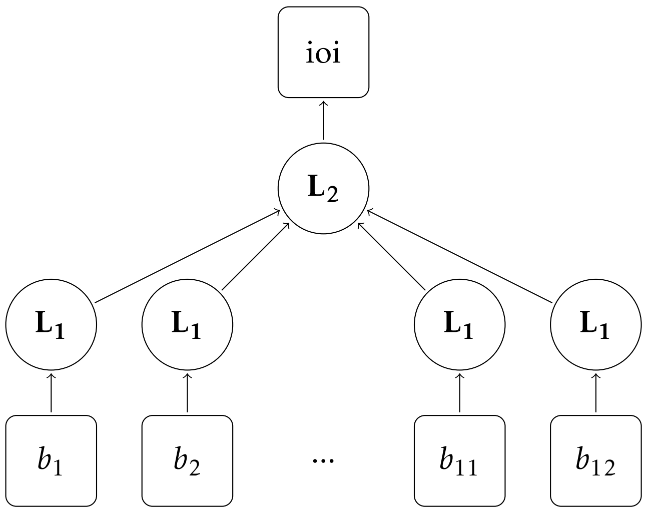


Figure 3.8 Diagram of the architecture used for tempo prediction. Hidden unit activations are sigmoidal, the top layer is linear (and fit using least-squares). $\mathbf{L}_1$ units share weights, an N-gram style setup is used.

not experiment with different variants of the architecture. However, in some experiments we use variations on the least-squares approach, using ridge and lasso regression (Tibshirani 1996) – the difference here is that instead of just minimizing the square error, we minimize the square error plus a regularization term that serves to keep the weights low. This somewhat counteracts overfitting, preventing weights with very high or low values used to fit particular details in the training data that might not reflect trends in more general data (see Section 2.2.9).

3.3 Implementation

The whole system was implemented in Python, with a strong reliance on the symbolic math library Theano (Bergstra et al. 2010). This library is especially suited for implementing neural networks and RBMs – computations are coded at a high level of abstraction, and are then compiled to machine code which is in many cases optimized to run on the GPU. The matrix computations required for these networks can then be strongly parallelized, resulting in significant speed-ups compared to CPU execution. The library also optimizes the computations – it replaces many complicated expressions by equivalent simpler ones.

The only unconventional functionality in our RBM implementation is the sparsity mechanism suggested by Goh et al. (2010). The precise implementation was slightly different from that described in the paper, to account for the way the RBM is coded in Theano, but it is mathematically equivalent (and therefore does not significantly affect the results in this thesis): in the method described by Goh et al. (2010) biased hidden unit *activations* are calculated explicitly, and are used to change the actual hidden unit activations. In our implementation, biased hidden unit *inputs* are calculated instead – this is due to the fact that in the Theano RBM implementation the gradient of the energy function is derived by the math library, while in other implementations an equation describing this gradient is supplied by the programmer directly. With our adapted procedure, the same gradient derivation can be used for the latent bias, and an interpolation between the normal gradient and the gradient of the biased activations is used for weight updates.

The feedback model (Section 3.1.2) causes one further complication. Normally, when we use neural networks for training or prediction we can process a large amount of data simultaneously, thereby benefitting from fast parallel algorithms (particularly when using the GPU). In the feedback model this is not possible, the features used as input for notes depend on the predictions for foregoing notes – velocity for notes is predicted in the order in which they occur in the score, and the predicted velocity is then used to generate the velocity-history sample for the next note. This considerably slows down the process, because parallelization can no longer be used. However, execution times are still sufficient for experimental purposes (although at this point, it would not be fast enough for real-time rendering).

3.4 Performance evaluation

After constructing and training a model, it is vital to test whether the model's predictions carry any meaning. We will now cover how the performance of the model is quantified – how we determine to what extent predictions of expressive parameters correspond to observed expressive parameters in actual performances.

The data we use for training and testing is highly heterogeneous. Although the expressive style seems to be fairly consistent within a piece, there is a lot of variation between pieces. This means that if we use only a single piece for testing, the results might not be representative (at all) for the performance of the model on the musical data in general.

The way we deal with this is by using a leave-one-out approach. When testing the model on a dataset, we train the model many times – every piece is used as test data once. So for the Chopin data, we first use Opus 1 as the test data and train on everything except Opus 1. This gives us one set of results. We then take the next Opus from the data, train the system on everything except that Opus, and evaluate the performance using this Opus as test data. This gives us another set of results, and this process is repeated, so that all data is at some point used to evaluate the performance of the model.

After the performance has been evaluated in this manner for every subset of the data in a particular experiment, the results are combined to determine a final overall performance of the model for this particular experiment, which is representative for the entire dataset. The weighted average performance measure $\bar{R^2}$ is calculated as follows:

$$\bar{R^2} = \sum_j \frac{R_j^2 \times N_j}{N_{all}} \tag{3.4}$$

where R^2 is the performance measure, N_j is the number of notes in piece j, and N_{all} is the total number of notes in all data.

The R^2 measure is the coefficient of determination, which is defined as follows:

$$R^2 = 1 - \frac{\sum_i (x_i - \hat{x}_i)^2}{\sum_i (x_i - \bar{x})^2} \tag{3.5}$$

Here x_i is the true target value for a particular sample, $\hat{x}_i$ is the prediction of the target value for the sample generated by the model, and $\bar{x}$ is the average true value over all samples in the piece. In more intuitive terms, the value represents how much of the sample variance is explained by the model (1 means perfect prediction, 0 or lower means no predictive value on average). In our model target values such as v_{N1} or Δt take the place of x in the expression.

Chapter 4

Dynamics

The two most important expressive parameters in solo piano music are the loudness with which notes are played (dynamics) and tempo/timing. A third parameter available to pianists is the use of pedals, but this is not further considered in this thesis. This chapter describes the experiments performed to predict expressive dynamics.

4.1 Experiments

A number of different experiments were done, related to the way the model was trained. Many of the variations on training were combined, e.g. experimenting with all different combinations of network set-ups and normalization styles. The unsupervised learning part forms a shared component, for which the results were used for all other experiments.

1. **Feature learning**: We only went through the unsupervised learning phase once. All available piano score data were used for this, and only one set of trained network weights for $\mathbf{L_1}$, $\mathbf{L_2}$ and $\mathbf{L_3}$ were used for all other experiments. The data we used for this are the JSB Chorales, Piano MIDI data and the Magaloff (Chopin) and Batik (Mozart) performance data. For more information about the datasets see Appendix A. These datasets together contain around 1.5 million notes, so 1.5 million samples were used for unsupervised learning. We see the unsupervised learning phase as an experiment because it is interesting to see what kind of features the model learns: we hope these relate to musicological concepts.

2. **Network configurations**: We experiment with different subsets of the full note-by-note architecture as described in the previous chapter. We use different combinations of the layers $\mathbf{L_1}$ through $\mathbf{L_3}$, and also test a model where predictions are fed back into the model to generate velocity-history samples for the piece at hand (we refer to this model as **VHF**). The reason we would want to train using a subset of the hidden unit

activations is that in this way we can get some insight into how important the different layers are for predicting performance parameters, i.e. how much information the different sets of feature activations contain relative to each other.

3. **Training datasets**: A number of different experiments were done based on the data, combining different sets of data for use as training or validation. In all cases, during supervised learning clearly target data is necessary (performance data, not just musical score). This is only available for the Magaloff and Batik performance datasets.

   (a) We trained on the Magaloff and Batik datasets separately, testing on the same dataset that was trained on in the leave-one-out fashion. This allows us to compare results to those obtained by Grachten & Krebs (2014) and Grachten & Widmer (2011).

   (b) We did experiments where we trained on one of the datasets (Magaloff and Batik), and tested on the other. Of course the datasets have a different nature (Romantic vs Classical period music, different pianists) so a performance drop is expected, but by doing this, one can inspect to what extent patterns learned from one dataset generalize when predicting performances for the other.

   (c) For the Magaloff data we also trained the system on subsets based on different genres of music that have been identified within the corpus (see Appendix A). In one set of experiments, we used these genres both for training and testing in the leave-one-out fashion. To be able to compare these results, we also trained on the full Magaloff dataset, testing on data only within the different genres. If the performance of the model trained only on data in one genre is better than that when trained on the full dataset, this suggests that some genre-specific characteristics are being modeled.

4. **Velocity normalization**: As described in Section 3.1.1, different methods of velocity normalization were applied, N1 to N3. The goal of these experiments is to try and isolate the structure in the velocity variations in different ways, and see if this affects how well the model can describe the variations.

4.2 Results

This section summarizes the results obtained with the described model.

4.2.1 Feature learning

Figure 4.1 shows the features learned from data in the note-centered representation (by $\mathbf{L_1}$). Many of the features look qualitatively similar, although there are a few types that seem common. Figure 4.2 shows a selection of features that illustrates some different styles of features that can be identified.

Some features show horizontal patterns (a), where interestingly the lines are exactly 12 pixels apart (corresponding to one octave in the input representation). In the example the lines are dark, meaning this feature is inhibited by a certain pitch class relative to the central one. This suggests that the model has recognized patterns relating to harmony or con/dissonance.

A group of features that is particularly prevalent has a thick dark band either in the top of the feature, or in the bottom (b). These features seem to react to notes that are in a certain part of the band of pitches that is used in the music – the illustrated feature would react to the notes with the lowest pitch sounding at a certain time, in other words the bass line. If the dark band is in the upper half of the feature, it will react to notes with a relatively high pitch compared to the context, i.e. the melody.

There are also quite some features showing vertical banding (c). This can be related to the pace of the piece (if the main duration for notes in a particular section is one quarter/eighth/sixteenth), and also to the relative position of the central note (i.e. is it on the beat grid, or offset).

Although most features exhibit either horizontal or vertical banding, there are a few that seem to suggest diagonal patterns (d). These features would be turned on when the central note is in an ascending (or in other cases, descending) sequence.

Finally, a number of features seem to respond to notes at particular locations in the grid (e). This could give some information about steps or trends in the melody, and about the harmonic role of the central note. Still, for all these features, one should be careful not to read too much into this, the exact mechanisms encoded in the neural network are hard to identify.

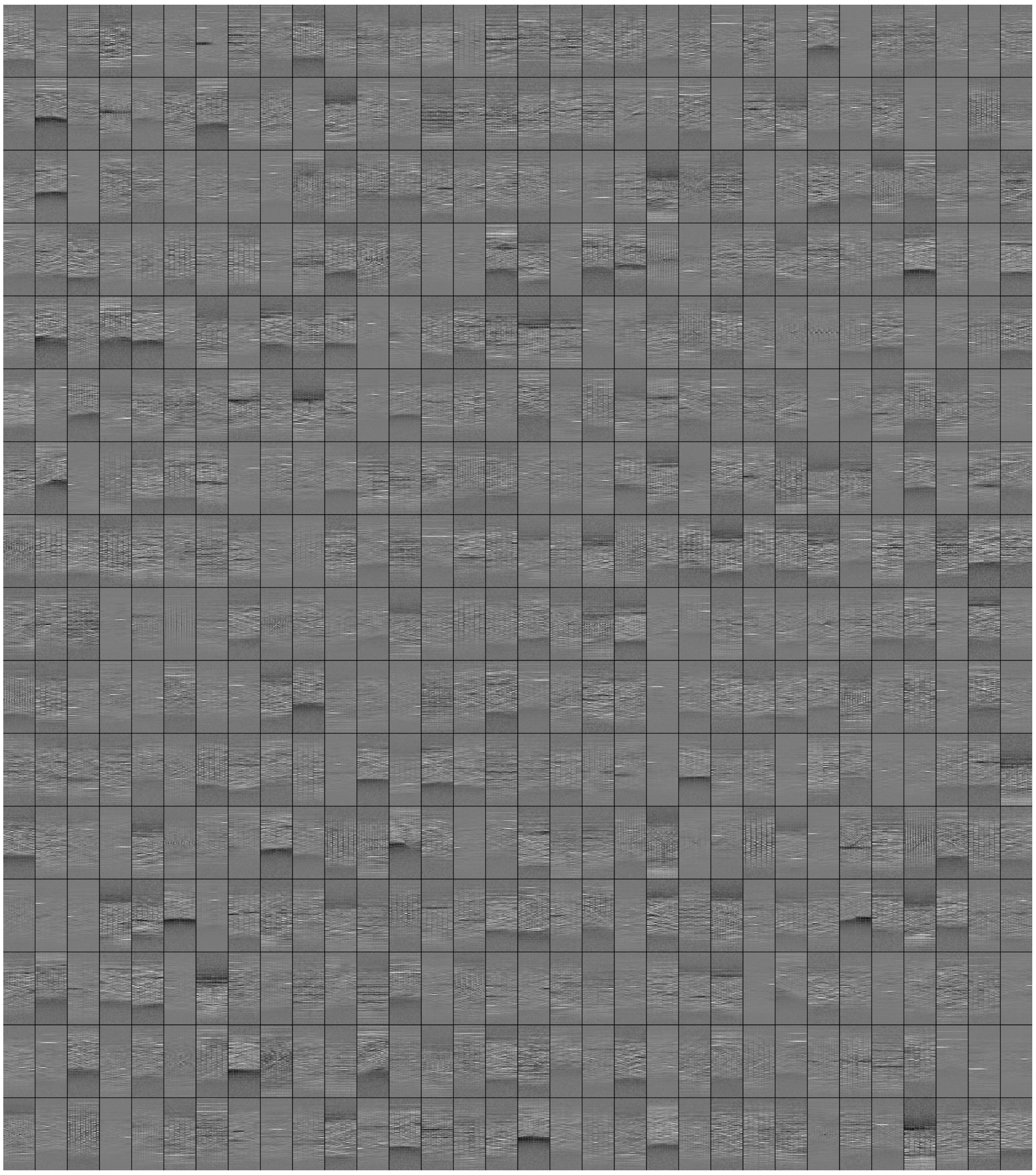


Figure 4.1 Visualization of a set of features learned on the note-centered representation. Light values correspond to positive weights, dark values to negative weights.

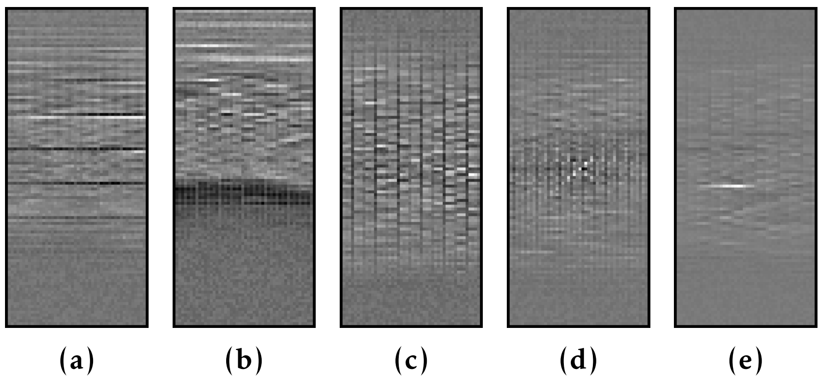


Figure 4.2 Some hand-selected features that seem representative for the types of patterns the system has learned. Dark values correspond to negative weights, light values to positive weights. Dimensioning of the features (necessarily) corresponds exactly to that of the input samples (see Figure 4.3).

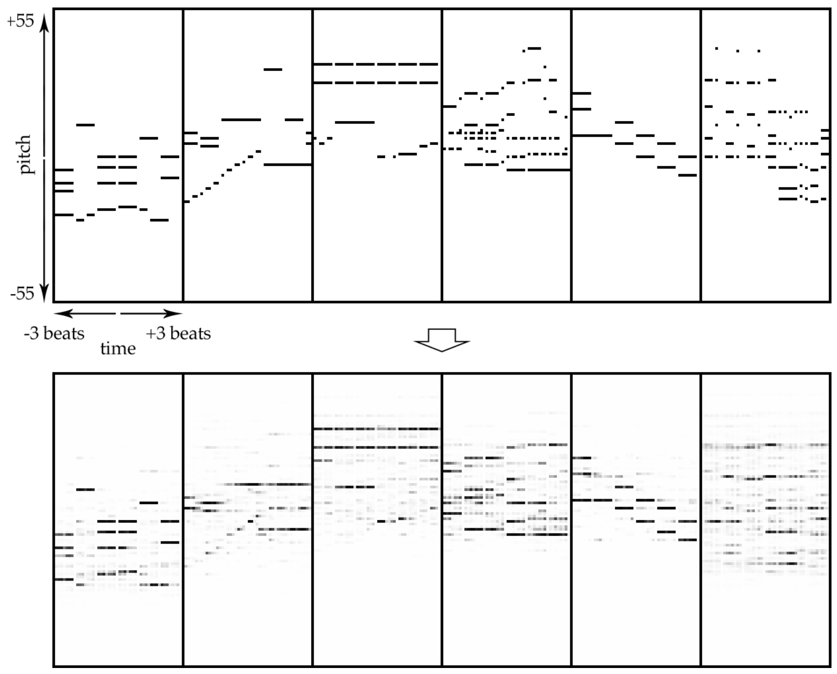


Figure 4.3 Some samples from the Magaloff data in the note-centered representation and their reconstructions with the learned feature bank.

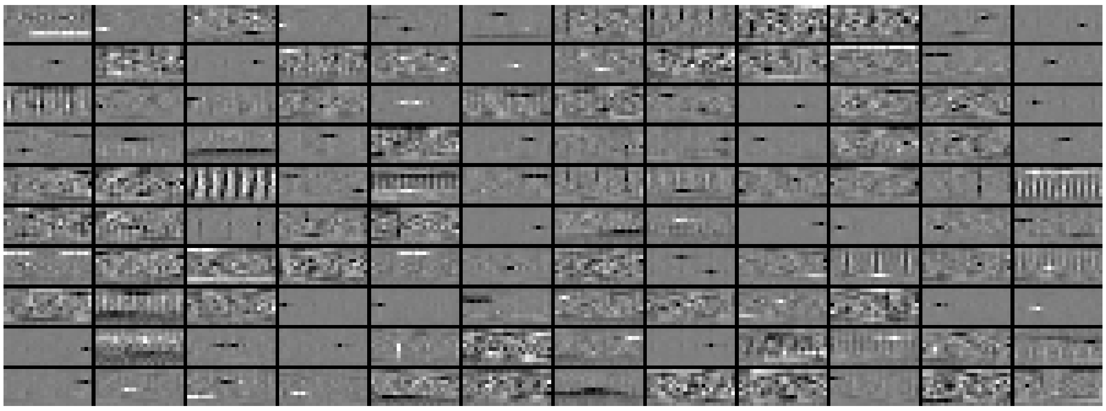


Figure 4.4 Visualization of a set of features learned on the velocity-history representation.

Another property of the feature bank that is interesting to investigate, is how well it can reconstruct the input data. The transformation applied by the neural network corresponds to a rather strong compression of the available data, and some information is generally lost, but with good features the loss is limited. Figure 4.3 illustrates some samples from the Magaloff/Chopin data, and their reconstruction from the hidden unit activations. Overall, the network does quite well: especially the first and the fifth of the samples are reconstructed quite accurately. However, in some places notes disappear (particularly notes with a short duration), and in some places the reconstruction actually contains notes that are not in the input (it would be interesting to inspect how well these extra notes would fit in the original music). The reconstructions can also be a bit blurry, and intricate details of a melody are generally lost. Still, most characteristics of the sample seem to be appropriately represented by the network, suggesting that the majority of information is retained as information flows through the network.

Figure 4.4 shows some of the features learned on data in the velocity-history representation. There seems to be quite a variety of patterns that it responds to: a number of features show vertical banding – this could relate to the pace of the piece or the relative position of the relevant note in the beat grid. Other features seem to react to notes at particular positions in the sample – this could be used to get an impression of whether the previous notes were played loudly, or whether the next note occurs a certain number of beats after the last loud or soft note. However, it is quite hard to say if the features correlate to 'phrase

	$\mathbf{L_1}$	$\mathbf{L_1L_2}$	$\mathbf{L_2}$	$\mathbf{L_3}$	$\mathbf{L_1L_2L_3}$	**VHF**
Magaloff						
N1	0.202	0.207	0.191	0.315	0.470	0.056
N2	0.196	0.200	0.184	0.304	0.459	0.030
N3	0.195	0.198	0.181	0.301	0.455	–
Batik						
N1	0.366	0.376	0.357	0.236	0.532	0.312
N2	0.358	0.368	0.349	0.229	0.523	0.300
N3	0.358	0.367	0.348	0.228	0.522	–
Batik → Mag. (N1)	0.132	0.126	0.125	0.286	0.386	-0.013
Mag. → Batik (N1)	0.291	0.295	0.283	0.209	0.457	0.175
All → Mag. (N1)	0.198	0.203	0.186	0.313	0.466	0.064
All → Batik (N1)	0.341	0.350	0.329	0.222	0.503	0.234

Table 4.1 R^2 scores obtained on the validation data. See Section 3.1.1 for a description of the different normalization strategies used (N1 to N3). The dataset indicated before the → indicates the set on which training was performed, and the dataset following it is the test dataset. In $\mathbf{L_1L_2L_3}$ velocity-history input is generated from an actual performance, in **VHF** this is generated through feedback of predicted velocity values – the latter experiments can only be done when the network generates N1 predictions (i.e. not with N3).

structure' or other important characteristics of the input data, and these observations are somewhat speculative.

4.2.2 Network configurations

Table 4.1 summarizes the R^2 scores achieved on velocity prediction for different configurations. It can be seen that in general, when looking only at score (so not using the velocity-history representation based $\mathbf{L_3}$ activations), around 0.2 of the velocity variance can be predicted by the model for the Magaloff data, and around 0.37 for the Batik data. When the velocity-history representation is added using the velocity data from the original performance, this score shoots up to close to around 0.5 for both datasets – clearly the context of the dynamics (as opposed to only the pitch) contains a lot of information, which is unsurprising: this way the model knows (for example) if it is in a section of the piece that is played loudly or quietly. Even the context of the dynamics by itself ($\mathbf{L_3}$) already contains a lot of information, more than just the pitch context.

When comparing the different score-only methods (using $\mathbf{L_1}$ and/or $\mathbf{L_2}$), it can be seen that using only $\mathbf{L_2}$ performs worse than using only $\mathbf{L_1}$ – this was also observed by Grachten & Krebs (2014), who concluded that stacked RBMs performed worse in general on this problem. However, when skip connections (connections between lower layers, in this case $\mathbf{L_1}$, and the output) are added, one can observe a slightly improved predictive coefficient compared to using only $\mathbf{L_1}$. Clearly there is some additional information contained in the $\mathbf{L_2}$ activations, although the improvement is small.

The **VHF** experiments performed worse than most other models. This was partly to be expected, because an error early in the process propagates in later predictions (see Section 3.1.1). There is a big difference in the performance of the model on Magaloff data (which is very poor), and the performance on Batik data (which performs slightly worse than the other score-based models in terms of R^2). This is likely to be related to the nature of these datasets: the Magaloff data contains large velocity variations, much larger than the Batik data.

4.2.3 Training datasets

In the experiments where we trained on the Magaloff data and tested on the Batik data and vice versa, only N1 was applied. Clearly, the cross-trained systems do not perform as well as the systems that were trained on the dataset itself - around 0.08 is lost from the R^2 score in all cases. However, the loss is actually not that large: the models do seem to have significant predictive value across the datasets. This suggests that the system has really learned something about musical expression in general, and not just about the context-specific aspects of expression, or about other patterns that hold only for one specific dataset.

It is also interesting to inspect the differences between the scores on Magaloff and Batik data. For example, it seems that the velocity-history representation makes a bigger difference to the prediction of Magaloff expression than that of Batik. This might be related to Romantic music relying more strongly on expressive phrases. On the other hand, using only score information seems

	L_1	L_1L_2	L_2	L_3	$L_1L_2L_3$
All data → genre					
Ballade	0.176	0.188	0.172	0.362	0.483
Etude	0.120	0.120	0.104	0.156	0.250
Impromptu	0.241	0.246	0.228	0.324	0.539
Mazurka	0.235	0.236	0.223	0.339	0.551
Nocturne	0.215	0.219	0.207	0.377	0.580
Piece	0.218	0.227	0.209	0.325	0.490
Polonaise	0.194	0.202	0.181	0.355	0.473
Prelude	0.132	0.132	0.132	0.234	0.369
Rondo	0.252	0.251	0.242	0.195	0.430
Scherzo	0.175	0.190	0.173	0.397	0.483
Sonata	0.175	0.175	0.157	0.320	0.435
Waltz	0.319	0.324	0.297	0.289	0.551
Genre → genre					
Ballade	0.128	0.135	0.146	0.355	0.450
Etude	0.069	0.056	0.079	0.160	0.203
Impromptu	-0.120	-0.135	0.060	0.265	0.280
Mazurka	0.246	0.242	0.239	0.364	0.555
Nocturne	0.216	0.204	0.209	0.383	0.579
Piece	0.170	0.160	0.173	0.309	0.449
Polonaise	0.085	0.074	0.122	0.349	0.413
Prelude	0.078	0.059	0.102	0.224	0.286
Rondo	0.189	0.149	0.212	0.171	0.340
Scherzo	0.090	0.065	0.133	0.397	0.437
Sonata	0.125	0.121	0.122	0.310	0.399
Waltz	0.307	0.293	0.304	0.339	0.541

Table 4.2 R^2 for velocity for different groupings of the training data. Scores that were improved by training on a subset of the data are underlined.

to result in better performance on the Batik data, suggesting that expression in Classical music might correspond more strongly to harmonic structure. However, these data are insufficient to draw definitive conclusions about these issues.

The results obtained when training and testing on the different genres in the Magaloff data are particularly interesting (Table 4.2). The top half of the table shows the results when predicting data from a certain genre using the model we trained on all Magaloff data. The bottom half shows the results that were obtained when predicting data from a certain genre using a model trained on that genre in leave-one-out fashion. The results where restricting the dataset to that specific genre was beneficial to the R^2 (raising it) are <u>underlined</u> in the table, and as you can see this occurs on several occasions. Although the improvement is small, this is an interesting result: a reduction of the amount of training data would not be expected to result in an increase in performance, rather the opposite. So even though the improvement is small, the improvement compared to what we would expect is quite notable.

The clearest improvements occur when testing the $\mathbf{L}_3$ architecture, suggesting that patterns in loudness sequences (as are encoded in the velocity-history representation) might be genre-specific to some extent, and (more interestingly) that the model seems to actually pick up on these patterns. This suggests that (a) the model encodes more than just average loudness in the note context (if this were the case the score would not improve for subsets) and (b) that if we would have more data in each genre, it might be possible to further improve these scores. This latter suggestion is strengthened by the fact that the largest genre in terms of both number of pieces and number of notes (mazurka, see Table A.1) outperforms the model trained on the full dataset in all configurations, and in general on genres containing more pieces and/or notes performance tends to be better.

The two cases where performance benefits most are on mazurka (+0.025) and waltz (+0.050) data. Interestingly, these are both traditional dances – only one of the other genres is a traditional dance (the polonaise), and this one looses a little bit of performance but is a rather small data set (6 pieces). Perhaps the patterns in loudness sequences encoded by the velocity-history model are relatively descriptive for dances: this is not unthinkable as they tend to have a repetitive and consistent structure in terms of accentuation.

The two cases where performance with the $\mathbf{L}_3$ architecture suffered most are the impromptu (-0.059) and rondo (-0.024) genres. These are at the same

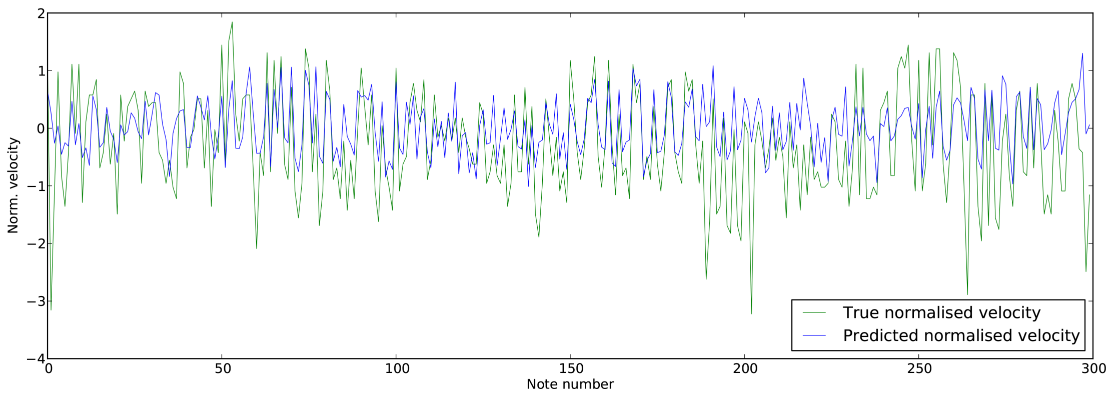


Figure 4.5 Predicted vs true note velocity for the first 300 notes of Chopin's Opus 1, using the $\mathbf{L_1 L_2}$ configuration.

time the genres with the smallest corresponding datasets (around 7000 notes and 17000 notes, respectively), suggesting the amount of available data might be part of the problem, and again suggesting that we could benefit from larger genre-specific datasets.

Finally, it is worth mentioning that we did some initial experiments using back-propagation (gradient descent instead of least-squares) for supervised learning with the genre-specific datasets, which suggested that the performance when training on a single genre could be further improved. This might be attributable to some overfitting when using least-squares (the genre-specific sets are a lot smaller than the full Magaloff dataset). When using gradient descent for the full dataset, we never obtained results as good as the least-squares results, but with experiments we did on specific genres, the performance on test data would in quite some cases improve by around 0.02-0.03. These results are not further discussed in this thesis.

4.2.4 Velocity normalization

When comparing the applied normalization strategies, strategies N1, N2 and N3 clearly rank performance-wise in that order. However, differences are minimal – particularly the difference between N2 and N3 is less than expected. Remember that Grachten & Krebs (2014) used N3, so improved results are partly attributable to the applied normalization strategies.

4.2.5 Significance of results

Although our average scores show an improvement over the results obtained by Grachten & Krebs (2014), we cannot simply assume that these results mean our model is better, particularly because there is quite a strong variation in the results between pieces. To more formally compare our results to those obtained earlier, we performed a paired t-test: this test can be used when measurements are repeated on the same subjects (in our case pieces) and the difference between the results on these subjects is normally distributed. We visualized the difference between the approach by Grachten & Krebs (2014) and ours, and the distribution indeed quite closely represents a normal bell curve.

Grachten & Krebs (2014) used exactly the same leave-one-out approach on the Magaloff data as we did, so we could use all our Magaloff results for a particular configuration. We chose to compare our best results in a predictive context to their best results. Their best results were obtained in a configuration similar to our $\mathbf{L_1}$ configuration with N3 normalization, with 1000 hidden units. We compared these to our $\mathbf{L_1L_2}$ set-up with N1 normalization (our best performing model on the Magaloff data when using only score data, no performance information). The mean improvement ($\mu = 0.078$, $\sigma = 0.08$, $N = 155$) was significantly greater than zero ($t(154) = 12.18$, two-tail $p = 2.4 \cdot 10^{-24}$) for more or less any typical α level (e.g. $p << 0.01$). We also determined the 95% confidence interval of the mean improvement to be (0.065, 0.091).

4.2.6 Other observations

Figure 4.5 shows a plot of the predicted velocity compared to the actual performance. Results are for the first 300 notes of Chopin's Opus 1, where the model is trained on all Magaloff data except Opus 1 using the $\mathbf{L_1L_2}$ architecture. The model seems to do quite well predicting peaks: qualitatively the curves seem to agree quite well, although the exact values are generally somewhat off.

The predictions made by the system seem to be rather conservative: the variation of the predicted velocity is visibly less than that of the ground truth. In fact, we determined that the standard deviation of the true normalized velocities for this piece is more than twice as high as the standard deviation of the prediction. To some extent, this is to be expected: the network optimizes for a high R^2 value, and can generally do better when its estimates are relatively safe (if it predicts very high and low values at the wrong locations, the R^2 score suffers strongly). However, in terms of rendering performances, a conservative prediction can be rather dull – when using the network to generate a rendering, this does mean it might be necessary to 'boost' the variation in velocity somewhat, for example by multiplying by a factor, where this factor can be chosen by hand or set such that the standard deviation of the output data is more comparable to that of reference data.

MIDI files were also rendered based on the predictions generated by the model. We rendered music using the $\mathbf{L_1L_2}$ architecture, and using the **VHF** setup. Although in both cases it can clearly be heard that the piece is not played by a (skilled) human, it does sound a lot more natural than a deadpan rendering. For rendering purposes, it was actually found that using the average prediction of the two models resulted in the most pleasing performances – $\mathbf{L_1L_2}$ by itself lacks some dynamism, while **VHF** sometimes seems to sometimes render less appropriate expression. The average seems to be a nice trade-off, but no listening studies have been performed, so it is too early for definitive conclusions.

Whereas earlier renderings using the system by Grachten & Krebs (2014) seemed to mostly relate velocity to pitch, the current model seems to take into account more of the context, it sounds more natural. This observation seems reasonable considering the higher achieved R^2 scores. The system still has a tendency to use pitch as one of the predictors for velocity, but this is to be expected and is in fact desirable: there is a correlation between pitch and performed loudness (Grachten & Widmer 2012).

4.3 Discussion

It is interesting to compare our results to those obtained earlier in similar settings, with the same dataset. Unfortunately, due to the (proprietary) nature of the datasets, there is not much literature we can compare to, but Grachten & Widmer (2011) as well as Grachten & Krebs (2014) have done similar experiments before. If we take their experiments with similar settings (using leave-one-out on the full set of Magaloff data), our results compare favorably.

Grachten & Widmer (2011) report an R^2 of 0.188, where their predictions are based on score annotations and absolute note pitch rather than the structure of the score itself. As these properties are quite distinct from the input that we use (a piano roll representation), it might be possible to combine these approaches in a hybrid model. There is a strong chance that such a hybrid model could perform better again than our model or that by Grachten & Widmer (2011).

Grachten & Krebs (2014) of course applied an approach very similar to the one in this report. In general, they achieve an R^2 of around 0.1, with a best obtained score of around 0.14. This is obtained with a network configuration equivalent to our $\mathbf{L_1}$ set-up, with 1000 hidden units and velocity normalization N3. As stated before, our results in a predictive context show a statistically significant improvement over these earlier results.

Although our predictive score on Magaloff data of 0.207 might still seem like a low score, it is important to put this number in perspective. As noted before, one very important omission of the model is the fact that it does not take into account score annotations. A large part of the expression in piano performances is presumably based on these annotations, as composers have a rather clear idea of how they imagine their composition to be played. Part of the creativity of the composer can actually be in these annotations, using annotations that are non-obvious for dramatic effect (which would be particularly hard to pick up on with a machine learning system that has no access to the annotations).

There are also many other factors that influence a particular performance of a particular piece: the pianist (his/her training, mood, character, etc), the composer, the musical style, the occasion, and many more. Because there are so many variables, it would be unrealistic to expect a system to obtain an R^2 (predictive coefficient) of 1, based only on performances of other pieces. In fact, the obtained 0.207 is not a bad score when taking into account all these factors – an interesting comparison would be to have Nikita Magaloff play the same piece twice, and compare how well one performance functions as a predictor for the next performance of the same piece - the corresponding R^2 would never be 1: even if a performer has a 'perfect' performance in mind, this does not at all guarantee that this perfect performance is in fact played, errors always creep in to some extent. However, it is hard to say how far from 1 it would be in practice, this would be an interesting subject for experimentation.

In the **VHF** experiment, the score obtained on Magaloff data is even much lower than 0.207. However, again it is important to put this into perspective. The goal of the model was not necessarily to accurately predict an original performance by one of the pianists: if the model does achieve this, that suggests that there is truth to the model, but if it does not, that doesn't make the model useless. It might still generate a viable rendering of the score, but just a different interpretation than the recorded one. This makes it important to also judge the results in a qualitative rather than quantitative way: one should consider whether what the model learned makes sense in a musical context, and whether it simply sounds good. In that sense, the model still performs alright (although the **VHF** model does have some clear weaknesses). Another thing to note is that the performance value is an average over the entire piece: the model might have correctly identified a number of patterns, but wrongly identified some other patterns which results in a low overall R^2 value.

Chapter 5

Tempo and timing

Having covered dynamics, it is now time to move on to the other important component of expression in piano music: tempo and timing. Tempo and timing are used by both composers and performers to affect the feeling conveyed by their music, and there is a strong interaction between what a listener expects and what happens in the music (sometimes expectations are met, sometimes they are intentionally exaggerated or violated). This chapter describes the experiments done, using models for predicting tempo and timing from the available performance data, and the results obtained from these experiments.

5.1 Experiments

Just as with the dynamics experiments, the tempo and timing models were trained in several different ways. For timing, as it also used the note-by-note model, experiments were mostly the same as the dynamics experiments and we refer to Section 4.1 for their detailed descriptions. For tempo, we did the following experiments:

1. **Feature learning**: Just as with dynamics, we only perform unsupervised learning once. This time, additionally score data from the Mutopia project (The Mutopia Project 2013), the Voluntocracy (Jaffer 2011) and Nottingham (Foxley 2001) databases, and a set of MIDI-files from MuseScore (MuseScore BVBA 2013) were used in the unsupervised learning phase (see Appendix A). Because of the different representation used for learning tempo, based on a sample per beat rather than a sample per note, the total number of samples was again roughly 1.5 million even with the extra data.

2. **Training datasets**: Because the number of samples for a given piece of score is lower when generating the tempo samples (there are typically much fewer beats than notes in a piece), no experiments were done on subsets of the performance data, as the resulting datasets would be too

small. Still, we used a few different sets of input, similarly to what was done for dynamics:

(a) We trained and tested on the Magaloff (Chopin) and Batik (Mozart) data separately, using the leave-one-out approach.

(b) We trained on all data, and then tested on one of the Magaloff/Batik datasets.

(c) We did cross-training, training on one of the Magaloff/Batik datasets, and testing on the other.

3. **Weight regularization**: For the tempo experiments, there were some issues with overfitting due to small training datasets (resulting in negative R^2 values). This is partly worsened by the application of a least-squares fit on the top layer when the input datasets are relatively small. To counteract this, the same experiments were done with ridge regression and lasso regression (Tibshirani 1996), where a regularization parameter prevented large weights (see Section 2.2.9).

5.2 Tempo results

Following are the results obtained from the described experiments.

5.2.1 Feature learning

Figure 5.1 shows some representative features learned by the model during the unsupervised phase. It seems the model mostly learned harmonic combinations: features are mostly bands at different pitches. There seems to be little structure along the time-axis, it has not integrated many patterns related to the pace of a piece (note lengths). Also detail seems to be limited to the central part of the pitch spectrum.

All in all, the features are quite rough, and visually the system does not seem to have learned very diverse or descriptive features. When looking at a reconstruction generated by the first layer in the architecture (Figure 5.2), the reconstruction loses detail where there are short notes, and in the lower and higher parts of the spectrum. Unfortunately it is difficult to visualize the $\mathbf{L}_2$ features in a meaningful way, if the model contains information about temporal structure that is where it should be expected.

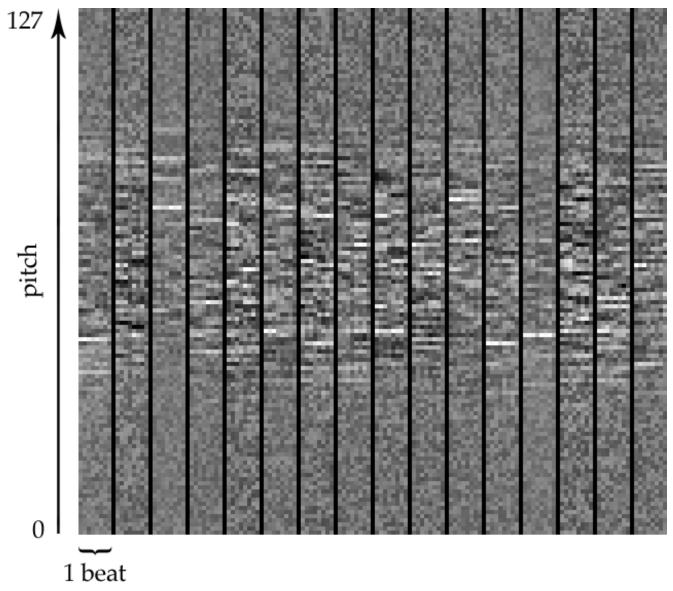


Figure 5.1 A subset of the features trained on the score representation. Light pixels correspond to positive weights, dark pixels to negative weights.

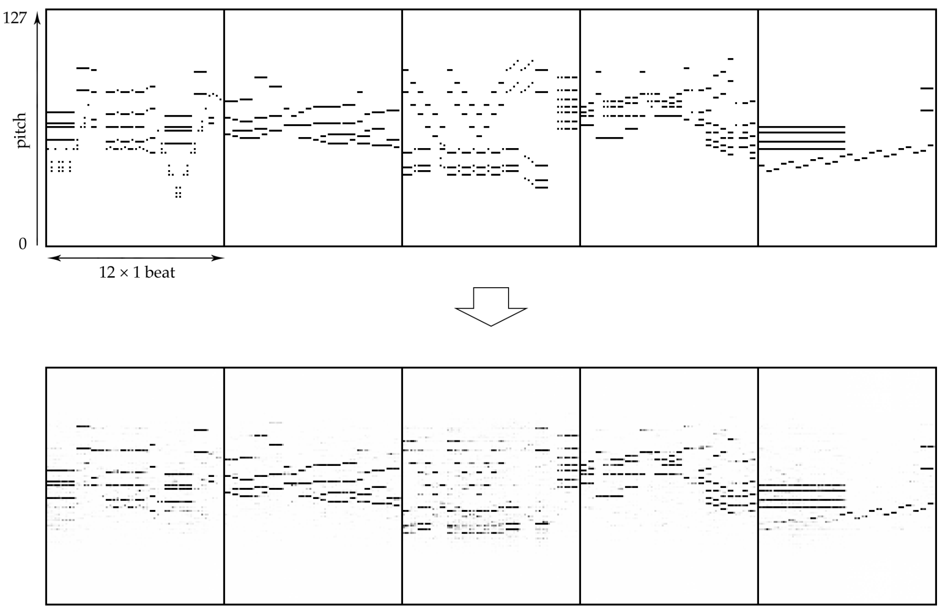


Figure 5.2 Reconstruction of 12-beat segments of input score using $\mathbf{L_1}$.

	Lst-squares	**Ridge** ($\alpha = 1$)	**Lasso** ($\alpha = 0.0001$)
Magaloff	-0.049	-0.042	-0.037
Batik	$-1.27 \cdot 10^{11}$	-0.553	-0.543
All → Mag.	-0.035	-0.039	-0.028
All → Batik	-0.287	-0.381	-0.25

Table 5.1 Tempo prediction R^2 results for different datasets and regularisation strategies.

5.2.2 Training datasets

As can be seen in Table 5.1, most experimentation using the tempo architecture was unsuccessful. One reason is that the available data is quite limited ($\sim$ 75.000 points of Magaloff data, $\sim$ 25.000 points of Batik data). This results in overfitting, which is in some cases quite bad: the learned weights generalize poorly to the test data, and in none of the experiments positive R^2 values were obtained. When training on small sets of data the problem becomes quite extreme, resulting in large negative R^2 values, particularly in the experiment on Batik data with no weight regularization.

The experiments were only somewhat successful on the Magaloff dataset, which contains the more interesting patterns with regard to tempo – the standard deviation of Magaloff tempo target values is roughly 3 times as large as that of the Batik data, indicating a very large difference in tempo variability. Still, when training on the Magaloff data, and validating using the same data, an average R^2 score of -0.049 was achieved – this is not a good score at all.

What is interesting, is that the score could actually be improved by including the Batik data in the training data (this is not the case in the dynamics experiments): even though the data is very different in nature from the Magaloff data, it results in an improved R^2 of -0.035 when validating on Magaloff data. When training on the full dataset and validating on the Batik data, an R^2 of -0.287 was obtained. Again this is can for a large part be attributed to overfitting, but it can also partly be attributed to the fact that the system was mostly trained on data that is not representative for the Batik dataset. When using only Batik data for training the dataset is much too small, and no reasonable results are obtained.

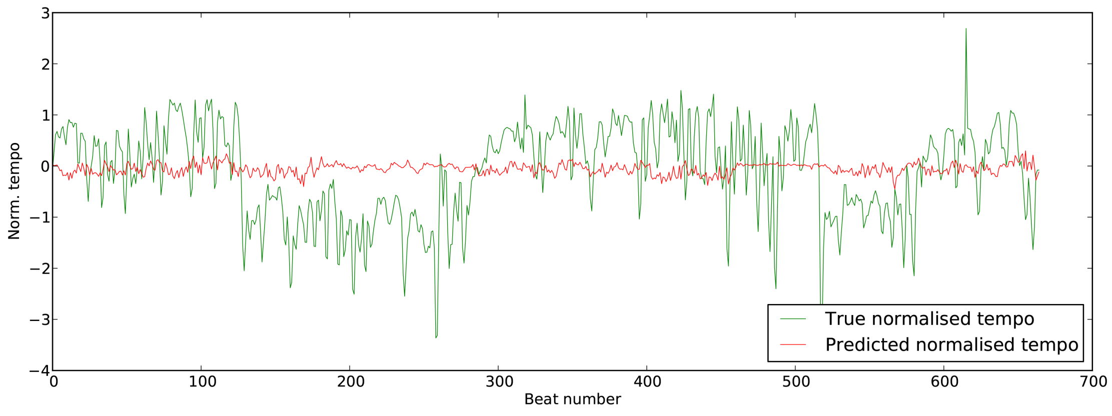


Figure 5.3 Predicted (no regularization) vs true tempo for Chopin's Opus 1.

5.2.3 Weight regularization

To counteract overfitting, the experiments were repeated with weight regularization. This reduced the problems somewhat: in all cases the resulting R^2 score became less negative. However, it is important to be careful with this: with extreme weight regularization, the system would actually start predicting 0's consistently, which would also result in an R^2 of 0, which is still better than our negative R^2. Fortunately, this does not seem to be the case when inspecting the predicted tempo data.

5.2.4 Other observations

Another factor limiting the success of tempo prediction, is that for many pieces, different parts of the piece are annotated at different tempi. However, because the learning system does not use any of the annotations, this means it will not recognize the parts that are supposed to be played more quickly or slowly, and therefore it will always tend toward a safe average. It therefore completely fails to follow the large-scale trends. This is visible in Figure 5.3, which graphs the performed tempo for Chopin's opus 1 (the green line, which starts high, then drops and later is consistently high again), and the predicted tempo. Another notable feature of the figure is that the absolute predicted tempo values are much smaller than the original values (red line) – the system predicts a safe average with few deviations.

Some other properties of the prediction are also interesting to note. One is that even though the predictions often do not agree with the available performance data, the predictions are relatively consistent: transitions between slower and faster sections are relatively smooth, and the tempo graph does not display strange peaks or jaggedness. This means that the model does treat slightly shifted versions of the same score in a very similar way (i.e. when the input score is shifted a few beats, the tempo prediction changes only a little), which is actually an interesting result, because this behavior is non-obvious.

Just as for dynamics, MIDI renderings were generated from the tempo predictions. Although it is hard to quantitatively say anything about these renderings just by listening, some qualitative observations can be made (although they are inherently subjective and, perhaps, optimistic). As was mentioned in the previous section, the predicted tempo curves seem surprisingly consistent: transitions are audibly smoother than would be expected, and in some cases they seem to be predicted at reasonable points in the score. In the beginning of pieces, the system seems to have learned something particular: it tends to predict a high tempo during the first few beats of a piece of music. Even though this beginning sounds a bit unnatural, in general the tempo predictions do not seem as bad as is suggested by the negative R^2 value.

5.3 Timing results

Prediction on timing was a bit better than for tempo, but still poor. In Table 5.2 the results of the timing experiments using different network configurations are illustrated (remember that the layer designations refer to the note-by-note architecture again). $\mathbf{L_1}$ has some features that seem to have minor correlation to the target values, adding layer $\mathbf{L_2}$ does not seem to have a beneficiary effect anymore. Interestingly, there seems to be practically no correlation between the velocity-history features and the timing data. All in all, only around 1% of the variation in timing can be explained using the model, which is very little.

Table 5.3 shows the results of separating the Magaloff data by genre, where only $\mathbf{L_0}$ is used as input. The first column of numbers shows the validation results on the different genres when training is performed on the full dataset,

	$\mathbf{L_1}$	$\mathbf{L_1L_2}$	$\mathbf{L_2}$	$\mathbf{L_3}$	$\mathbf{L_1L_2L_3}$
Magaloff	0.014	0.013	0.012	0.002	0.016
Batik	0.013	0.010	0.011	0.000	0.012

Table 5.2 R^2 scores for timing obtained on full datasets using different training architectures.

Genre	All data →	Genre →
Ballade	0.016	-0.025
Etude	0.000	-0.074
Impromptu	0.014	-0.295
Mazurka	0.023	0.012
Nocturne	0.018	-0.023
Piece	0.007	-0.048
Polonaise	0.019	-0.039
Prelude	0.024	-0.102
Rondo	0.013	-0.107
Scherzo	-0.008	-0.113
Sonata	0.008	-0.042
Waltz	0.063	0.045

Table 5.3 R^2 scores for timing for different groupings of the training data ($\mathbf{L_0}$).

the second column when training only within the genre used for testing. Some genres seem to display somewhat more recognizable structure in the timing than others, particularly in the waltz genre the score is a lot higher than for all others, in both training scenarios. The same holds to a lesser extent for the mazurka genre.

5.4 Discussion

The applied architecture for tempo is somewhat limited in its capabilities. It is very rigid in the sense that it always gets score input of the same length as input, and score is represented at its original pitch. This means it cannot adapt flexibly to phrases or sections of varying lengths, or situations where in some cases patterns occur in the form of quarter notes, and in some cases in eighth notes. It also cannot learn a flexible interpretation of harmony: to recognize the same pattern in different keys it would have to learn the pattern in each key separately (it is not insensitive to absolute pitch, whereas most people are).

At the same time, these characteristics have some advantages: in theory it is a suitable system for recognizing harmonic transitions (because different keys are represented in different features, a clear transition from one set of features to another can be observed). It should also be suitable for determining the meter of a piece (e.g. if activated features for the first beat are very similar to those of the fourth, this is a strong indication of a 3/4 or 6/8 meter).

It might be possible to deal with the issue of pitch sensitivity in different ways, one is making the model pitch insensitive (this idea is developed in Appendix B), another is using harmonized data where the known harmonic properties are taken into account in the model. This was for example done by Boulanger-Lewandowski et al. (2012), where they offset their input data to always start with C as the root note. Finally, one could artificially enlarge the input dataset by including pitch-shifted versions of all the available data – this would ensure that if a pattern occurs in a piece in one key, it is also seen by the learning system in other keys.

Using the regularized regression methods for training on the tempo data (ridge and lasso regression) improved the tempo results. This seems to be related to somewhat better generalization by the model. However, the problem that there is too little data available cannot be fixed by using weight regularization, the technique can only be used for damage control. It seems like we have several problems: (a) too little data; (b) an ineffective model; and (c) possible complications in the available performance data with respect to onset times and/or the way we transform the data into tempo/timing values.

Finally, something to note about the R^2 value is that it is an aggregate value for the whole piece: if this value is negative, it means on average that the model did quite badly. However, it could be the case that the model is quite accurate in some parts of the music, but very bad in other parts – this would still cause a negative R^2. Listening to tempo renderings seems to suggest that this might be happening, as the predictions are not consistently bad.

The timing results are better than the tempo result. The fact that predictions on the waltz genre are higher than in other genres is particularly interesting, because a strong timing effect is known to exist in waltz performances: of the three beats in each measure, the first is normally played shorter than the other two (Askenfelt 1986, Bengtsson & Gabrielsson 1977). As the mazurka is also a traditional dance, the slightly higher scores obtained in this data suggest that perhaps something similar is happening there.

Chapter 6

Conclusions and future work

In Section 6.1 we summarize the conclusions of the experiments in this thesis. A number of suggestions for future work are also made, some relating to possible improvements of the systems as they are now, and some to alternative systems that might perform better. These are covered in Section 6.2.

6.1 Conclusions

The objectives of this study were twofold: to develop a model for predicting expressive performance, and to discover and give insight to features that relate to expression. Grachten & Krebs (2014) took an initial step in this direction, and we hypothesized that we could improve on their results, as well as extend the method to different contexts, with a focus on Restricted Boltzmann Machines (RBMs).

To study dynamics and timing, we adapted the architecture developed by Grachten & Krebs (2014) in a number of ways. One important change was the step from a dense coding, where information is spread out over many nodes in the neural network, to a sparse coding, where every input sample is encoded in only a few nodes. Also, when using multiple layers of RBMs (stacking them), we encoded additional connections in the model allowing it to combine information from different sets of features at the same time. Finally, the note-centered representation was augmented with an additional velocity-history representation, encoding the short-term musical history with regard to loudness of the previous notes.

The architecture was trained on a larger amount of score data than was done previously, resulting in an improved model. Results obtained with the model are better than those obtained by both Grachten & Widmer (2011) and Grachten & Krebs (2014) on the Magaloff (Chopin) performance loudness data in a predictive context. The learned features also seem to relate more clearly to musical concepts: features seem to encode patterns with regard to rhythm, harmony (through relative pitch), and some basic sequences such as ascending and descending patterns. On another performance dataset containing piano music by Mozart, performed by Roland Batik, an even higher performance was observed when using our model.

The velocity-history representation also proved useful, as it could encode information about the expressive context of notes rather than only harmonic context. When explaining loudness of separate notes, considering the expression of previous notes as known (i.e. predicting the loudness of the next note in a sequence), the full model could predict a large amount of the variation in both the Magaloff and the Batik datasets. Part of this can be attributed to the fact that the model now knows whether it is in a loud or soft part of the piece, but this is not the only mechanism at play. When training the system on datasets that were more homogeneous in terms of musical style (consisting only of music of the same genre), the amount of information encoded by the velocity-history representation increased, suggesting that it enables the model to learn some genre-specific patterns (the improvement was particularly pronounced in the traditional dances mazurka and waltz).

When predicting velocity for a new piece, one cannot consider the expression of previous notes as known, since no information about expression is available. For this situation, a model was developed where predicted velocities where fed back into the system to generate the velocity-history samples for successive notes. Because here the history does not describe the true performance, which we use to judge the quality of the model, understandably the R^2 score drops significantly, and in fact the prediction in this manner performs worse than when leaving the velocity-history representation out of consideration.

To ensure that the model was actually learning about music in general, and did not just learn to recognize particular patterns in our dataset, we also performed cross-validation – training the model on one dataset, and testing it on another. Although performance suffered somewhat, this drop can be partly attributed to the fact that musical styles were not the same between the datasets, one containing music from the Romantic period, the other music from the Classical period, by different composers and performers. These results suggest that the model has really picked up on meaningful patterns in the performance data.

A different architecture was also developed, where the focus was not on expression of separate notes but on expression in particular time-spans of the music. Whereas the previous architecture parsed the music one note at a time, this architecture took in score 12 beats at a time, and was trained as an N-gram. This architecture was relatively rigid in terms of pitch, as it had a strong reliance on the absolute pitch of input data. Efforts to overcome this dependency using a wrap-around convolutional RBM (described in Appendix B) were somewhat successful, but the system was not mature enough to apply successfully to performance data.

Unfortunately, the results with the N-gram architecture were rather poor: almost none of the variation in tempo in the performance data could be explained or predicted, and the learned features did not seem to encode musically meaningful information. Part of this problem may be due to the fact that there was relatively little performance data available for training, and that there were some problems extracting meaningful tempo information from the performance data.

Experiments analyzing timing were also done, using the note-by-note architecture. Again, results were poor overall, suggesting that part of the problem when predicting tempo and timing might have been inherent to the data, or the way it was converted into tempo and timing values. Still, there was one scenario in which the model seemed somewhat successful: when predicting timing data for music in the waltz genre. The waltz is known to have a particular and consistent timing pattern (Askenfelt 1986, Bengtsson & Gabrielsson 1977), suggesting that the model might have learned something meaningful here.

6.2 Future work

Although some positive results have been obtained, of course there are still many opportunities for improving the system, or for accomplishing the same goal in a different manner. Section 6.2.1 suggests possible improvements, i.e. extensions to the current system. Other limitations are simply inherent to the chosen structure of the model: Section 6.2.2 suggests more advanced setups that might overcome some of these problems. Finally, Section 6.2.3 highlights some conceptual issues that relate to how humans learn about music, and how these reflect on designing machine learning systems that should accomplish the same.

6.2.1 Improving the current system

There are a number of features that could be added to the system described in this thesis that could make it perform better. The following sections describe some opportunities for further development, where the main structure of the system as it is now would remain in place, but its functionality is expanded or its performance is improved.

Score annotations One drawback of the current system is that score annotations are not taken into account. Grachten & Widmer (2011) showed that (unsurprisingly) a lot of information about the expressive content of music is contained in the score annotations, and in fact their obtained R^2 score (0.188) is very close to ours, while relying on a mostly different set of features. Although for this thesis the goal was mostly to look for structure in the bare score (only using pitch and duration), when using the system to render performances it could potentially benefit significantly from this extra information, and combining the annotation features used by Grachten & Widmer (2011) with the features described in this thesis could result in a model with a much better predictive capability than either of these models separately.

Additional data In many cases, when one wants to improve a machine learning system, a way to achieve this is to supply the system with more data. There are several good reasons why our system might benefit from more available data:

- For some experiments little data was available, which seemed to limit the performance of the model, in part due to possible over-fitting. This seemed problematic for the experiments on music from different genres, and for the experiments on tempo data.

- The model has been exposed to a quite specific set of data: the available performance data described performances of music by two composers, by two different pianists. This makes it hard to say to what extent the model generalizes to other musical styles, genres, performers, etc.

- A common trick to make machine learning systems insensitive to certain transformations on the input data, is to train the system on transformed versions of the input data as well as on the original data. This additional 'synthetic' data can also counteract over-fitting to some extent. In our case, we could attempt to train the system on additional pitch-shifted transformations of the input data to reduce sensitivity to absolute pitch.

Dealing with these factors would require different kinds of additional input data – to better train the model on a particular genre, data specifically for that genre would be needed. To train it on expression in jazz-piano (for example), specifically that kind of performance would need to be recorded. This kind of data is hard to obtain. There is however some additional data available that was not used in this study and that could prove useful, this is described in Section A.3 in the appendix.

Fortunately, dealing with the factor in the last bullet-point would be relatively straight-forward: it is not too complicated to simply shift the input data. However, one would need to take care: providing the system with pitch-transposed data implies a very strong assumption that a change in absolute pitch actually has no effect on the musical structure, or on the corresponding expression. Another thing to note is that this approach would only work for the beat-by-beat model: in the note-centered representation the input data is already insensitive to absolute pitch, and transposing the pitch of the input score would not actually change the way the model sees the data.

Model fine-tuning As was mentioned in Section 1.2, the experiments in this thesis focus mostly on comparing results for different styles and parameters, and generally improving the RBM architecture for this application. In that sense, the research objective was accomplished: some interesting results have been obtained for different input datasets and expressive parameters, and the RBM architecture seems to have been improved through the implementation of a sparsity mechanism, addition of skip connections between the lower layers and the prediction layer, and an improved normalization strategy.

However, Grachten & Krebs (2014) showed that there was still some variation in the performance of the system depending on some other parameters, such as the timespan of score used as input. In general, RBM's and neural networks also allow for tuning of many different hyper-parameters – among others the number of units in the network, learning rate, and sparsity and weight regularization parameters. For the results in this thesis, a 'satisficing' rather than optimizing approach was applied: when the learned features and predicted values were of sufficient quality, we focused on trying different things with the system rather than obtaining the maximum possible performance by optimizing all parameters. This means it might be possible to improve the performance of the system by finding a better training configuration, particularly if a thorough strategy for optimizing hyper-parameters is applied, such as one of the approaches suggested by Bergstra et al. (2011).

6.2.2 Opportunities for exploration

Although the existing system can be improved in different ways, the chosen architecture is not the only one that could work for this particular task. An advantage of the applied system is that learned features have a rather clear relationship with the input score: we can simply inspect the feature bank to see to what kind of patterns the system responds. However, the way the system is designed is also quite rigid: the time and pitch spans taken into account are fixed, so as soon as a piece of score moves out of this fixed context, it is no longer taken into account by the system.

There might be other systems with a better modeling capacity with regard to these issues. One way to deal with this would be to use recurrent neural networks instead of what is done now – although recurrent neural networks are hard to train, and very hard to inspect, they effectively have a more flexible 'memory'. If the system were redesigned as a recurrent neural network, the time-aspect of the input data would no longer be explicitly encoded in the input features (as it is now, using a time $\times$ pitch grid), but it would be included more implicitly as the network unfolds over time.

To enable the network to encode states that can truly store memories of historical events over a longer timespan, long short-term memory (Hochreiter & Schmidhuber 1997) could be used. To deal with the fact that in music multiple events can occur at the same time (whereas in text letters follow each other one at a time, in music multiple pitches can sound simultaneously) the same approach could be applied as was done by Boulanger-Lewandowski et al. (2012) in their RNN-RBM system. Finally, the recurrent network could be made even more capable by applying multiplicative connections trained with HF optimisation as was done by Sutskever, Martens & Hinton (2011). This last system was trained on large amounts of text from Wikipedia, and became quite able to generate text that looks natural – the system seemed to have a sense of the grammatical role of different words, and was able to match parentheses over long distances. This is a very promising basis for a system that should understand the syntax of music.

To reduce the sensitivity to absolute pitch of the input, it might be possible to take a convolutional approach. In a convolutional network, the network is evaluated at different positions in the input data. One architecture that was experimented with, but abandoned, during the work on this thesis used convolution in a wrap-around sense. This architecture is described in more detail in Appendix B, and might prove useful in certain scenarios.

6.2.3 The importance of starting small

It is very tempting to simply take a large amount of complicated data, and attach it to a machine learning system, expecting the system to find the relevant substructures and patterns. However, it might not be realistic to fully understand music this way – structures in music are complicated: there are hierarchical patterns but also strong relations over time.

In this sense, music is very similar to language, and interestingly it seems for a large part to be processed in very similar ways in the brain (Patel 2003). Todd (1985) also observes parallels between his model for timing in music and observations related to pauses during speech (Grosjean, Grosjean & Lane 1979). For language, it seems to be very important to start with understanding of simpler structures before moving on to more difficult structures, which is illustrated very elegantly in the paper *Learning and development in neural networks: the importance of starting small* by Elman (1993). This can be achieved by (somewhat counterintuitively) limiting network capacity, or by carefully determining the order in which data is presented to the system, i.e. curriculum learning (Bengio et al. 2009). When thinking about this issue, one should realize that for people there is also a tendency to start with simpler music during childhood (lullabies, music boxes, etc) and to later advance to more complicated music. In fact, some music is so complicated that people need to build their knowledge and understanding of music theory and history to be able to fully appreciate it.

In the experiments described in this thesis, during training there was no gradual progression from simple music to more complicated music whatsoever. Samples also contained a relatively large amount of information straight away, as in the note-centered representation pitch and rhythm were aggregated into a single input vector with no immediately apparent structure (when seen by the network). Perhaps, by training the system to understand harmony and rhythm separately first, and combining this understanding at a higher level of hierarchy, better results could be obtained. Experimentation with curriculum learning, where first the network learns from simple patterns (e.g. folk music), and later uses this knowledge to learn more effectively about complex music, might also prove useful.

Appendix A

Available data

A number of different datasets were available for this research. For the experiments a collection of MIDI files (Section A.1) was used in combination with performance data for pieces by Mozart and Chopin (Section A.2). Some other data was not used for this project but might be useful for future work (Section A.3).

A.1 MIDI/score data

MIDI data has been obtained from a number of sources:

- **JSB Chorales:** A dataset comprising 498 chorales by Johann Sebastian Bach and has been obtained from `http://jsbchorales.net` (Greentree 2011).
- **Musescore:** A set of 240 MIDI files from `http://musescore.org`
- **Mutopia:** A dataset of 4182 MIDI files with strongly varying characteristics. Many different musical styles are represented. It has been obtained from `http://mutopiaproject.org`.
- **Nottingham database:** This is a database of 1037 folk tunes. (Foxley 2001)
- **Piano-midi:** A dataset containing piano music from a large number of composes, in many different styles, from `http://piano-midi.de` containing 327 pieces in total. (Krueger 2013)
- **Voluntocracy:** 54 folk tunes from `http://voluntocracy.org`. (Jaffer 2011)

In total 6338 MIDI files are available but they have not been checked for duplicates. All MIDI data is strictly timed and all notes have a constant velocity (they are dead-pan renderings, so no performance information is available).

A.2 Detailed performance data

The OFAI (Österreichisches Forschungsinstitut für Artificial Intelligence) owns a number of proprietary data sets describing piano performances. The most detailed data sets describe piano music composed by Chopin as performed by Nikita Magaloff (~ 300.000 notes) and Mozart's piano sonatas as performed by Roland Batik (~ 100.000 notes). These have been recorded on a Bösendorfer piano, which has measured note on- and offset times and hammer velocities. Both data sets have been linked to the score, such that for every tone in the performance it is known to which note in the score it corresponds. Score annotations (about dynamics and tempo) are partly available in the dataset; not all annotations have been appropriately recognised in the automated analysis. The Magaloff/Chopin data describe music in different genres. The way the Magaloff data have been processed is described in more detail by Flossmann et al. (2010). The following sections describe some of the musical characteristics of the data.

A.2.1 Composers

Detailed performance data is available for music by Mozart and by Chopin. The main difference between the two is that Mozart is a composer from the Classical period of music, and Chopin from the Romantic. This section focuses more on the differences between these two periods, than on the characteristics of one or the other.

In the Classical period (~ 1750-1810), musical structure was more strict. Form was rather strictly adhered to and played an important role. Harmony was relatively constant over the course of a piece, with few changes in key or scale. Tempo could be different in different movements of a piece, but overall would be rather invariant.

Music in the Romantic period (~ 1810-1890) was more about telling a story, and this story-telling aspect reflected in several factors. Form was less strict, and many composers experimented with deviations from standard forms. There would also be more harmonic change during the course of a piece, causing changes in mood. Expanded chords were used more (more tones at the same time, and different combinations), and *tempo rubato* (Italian, lit: stolen time) played an important role: changes in tempo on a smaller timescale.

A.2.2 Genres

All Mozart's pieces for which detailed data relevant to this project are available, can be classified as *piano sonatas*. The term sonata refers to the fact that an instrument is of main interest in the piece, as opposed to a voice (in a *cantata*). However, the term also strongly relates to the musical form of the pieces: in general, the pieces would consist of a three or four movements, of which the first would be in *sonata allegro* form. This form would sometimes start with a slow introduction, followed by the exposition in which two themes would be presented. After the exposition, the themes are developed: complex variations of the theme are played. This is followed by the recapitulation (referring back to the themes), and sometimes a coda to close the movement. The other movements of the piece could be of different forms. (Sadie 1980)

The available piano music of Chopin can be classified in several groups, the largest of which are *etudes*, *mazurkas*, *nocturnes*, *preludes waltzes* and *ballades*. Table A.1 lists which genres are available, which opuses belong to the respective genres and how many pieces there are in total along with the total number of notes.

The main commonality between the etudes is that they are written as piano exercises, and are challenging to play. The mazurkas are based on a traditional Polish dance, and share some rhythmic features as well as patterns in their form.

The nocturnes

> "are typified by a tuneful and ornamented melody, with a left-hand accompaniment based on flat or broken chords. This tunefulness and the rich and refined ornamentation point to the vocal character of the melodic line, bringing to mind the *bel canto* Italian operatic style (Chopin was a great admirer of Bellini)." (Bielecki 2010)

The preludes are all short pieces, all shorter than 90 bars. With regard to musical form, they have little structure, and are in a way musical poems.

Chopin's waltzes are in 3/4 meter, as is typical for waltzes in general. However, they are not necessarily intended for dancing. The waltzes are quite varied in character, with 18 of them published of which 8 were originally intended by Chopin for publication, which were considerably larger in dimension than the others. Within these eight a division into two groups can be made again,

Genre	Opuses	# pieces	# notes
Ballade	23, 28, 47, 52	27	33136
Etude	10, 25	22	39105
Impromptu	29, 36, 51	3	7059
Mazurka	06, 07, 17, 24, 30, 33, 41, 50, 56, 59, 63	41	44886
Nocturne	09, 15, 27, 32, 37, 48, 55, 62	18	30016
Piece	12, 19, 43, 46, 49, 57, 60	8	32184
Polonaise	26, 40, 44, 61	6	26328
Prelude	28, 45	25	19428
Rondo	01, 05, 16	3	16986
Scherzo	20, 31, 39, 54	4	21137
Sonata	04, 35, 58	12	36879
Waltz	18, 34, 42, 64	8	17959

Table A.1 Opus numbers and their genres.

some of the virtuosic *valse brilliante* style, and some of a more melancholic and slow character. (Bielecki 2010)

The ballades are relatively large works (8 to 12 minutes in length) in triple time (6/4 or 6/8), which contain dramatic and contrasting subjects. Among the ballades, there are large differences and they cannot be easily classified as a coherent group (although they exhibit some similarities in form). Supposedly, even after pianists have mastered the technical difficulties in these pieces, they still consider the poetic interpretation a real challenge. (Tran 2009)

A.2.3 Pianists and performances

Roland Batik (1951) is an Austrian pianist, who focuses on jazz piano. However, he also played Mozart's piano sonatas, for which performance data is available. In 1991 he received the 'Wiener Flötenuhr' price for his recordings of the piano sonatas, being chosen as the best interpretation of the year.

Nikita Magaloff (1912-1992) was specialised in the works of Chopin, and was the first to record Chopin's complete works. When the described dataset was recorded, Magaloff was nearly 80 years old. His accuracy had by then decreased, and there are some musical errors in the data. Over the course of his life, his performances of Chopin also supposedly became more passionate and daring. The recordings might not be completely representative of the earlier performances (Gerhard Widmer, personal communication).

A.3 Other data

Beside the mentioned data, the OFAI owns a collection of beat-tracking data for classical performances. Music by a variety of composers, played by a variety of performers has been analysed to track rough tempo and loudness information over the course of the performances. This data is available at beat-level – loudness values are not available for individual notes and the data has not been as thoroughly matched to score as the data from Section A.2. In principle this data is usable with the beat-by-beat model described in Section 3.2, although this would require some work for aligning performance data and samples.

Performance data that might also be interesting to study in future work has been developed by Nettheim (2013). For a piece by Chopin, he transcribed three different performances by the same pianist, describing dynamics and timing at note-level. Although the fact that the transcription has been performed by hand raises some issues with regard to reproducibility (particularly for velocity estimates), this might allow the comparison of detailed data of multiple performances of the same piece by the same pianist – none of the other described sets contain this kind of data.

Appendix B

WARB Machine

In many cases, if patterns occur in different locations in input for a neural network (for example, small shifts in where a number is located in the image), qualitatively this does not strongly affect the meaning of the pattern (a shifted 1 is still a 1). In music this effect also arises: a melody heard twice at separate occasions with only a minor pitch offset (a few semitones) sounds very similar.

Unfortunately, RBM's and neural networks normally are very sensitive to small shifts: if a small shift in the input vector occurs, completely different hidden units can be activated – this could mean that a whole different part of the network is used, while one would want the mechanism to be insensitive to these minor variations. In neural networks, a way to deal with this is to apply a convolutional architecture: here the same network configuration is evaluated at different locations of the input vector. In this way the same 'knowledge' is valid at different locations. This has been applied quite successfully for hand-written digit recognition (LeCun et al 1995). For RBM's, convolution has not often been used, although it has been done (Norouzi, Ranjbar & Mori 2009). The structure of RBM's lends itself to convolution a bit less well than the structure of regular feed-forward neural networks.

There is another characteristic of music that seems quite important when trying to understand its structure: there is a strong similarity between pitches in different octaves which share the same pitch class. In a way, the pitch has a cyclical nature: a number of pitch classes repeat over and over again. Considering this fact, it might also be useful to try and train the system first on a simplified form of the music, where the system only observes the pitch classes that are in the music: this already contains a lot of information about harmony, and the system is not distracted by extra information on absolute pitch.

The combination of these characteristics (absolute pitch is of limited importance, and a lot of information is retained if the input is compressed to a single octave) led to some experimentation with an architecture that indeed

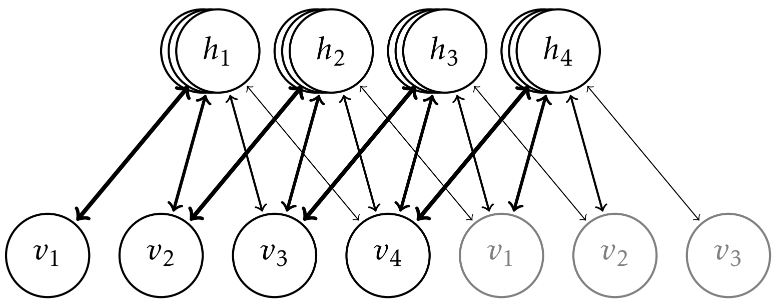


Figure B.1 An RBM with weight-sharing, such that the result is a wrap-around convolutional RBM. Arrows with equal thicknesses share weights, i.e. each of the hidden unit activations h_1 to h_4 are based on the same weights, but shifted. There can still be multiple sets of weights – if there are n_v visible units, and n_w features, there are effectively $n_v \times n_w$ hidden unit activations. In principle, this is the same as a normal RBM, but then evaluated at every possible offset simultaneously.

only observes pitch classes and their relative distances (it is insensitive to absolute position). However, the architecture was abandoned early on because its maturity was insufficient to be able to include it in a full-fledged system.

The implemented RBM uses a wrap-around form of convolution (where one edge of the input vector is considered connected to the other, i.e. the input vector is cyclical), and was dubbed WARB Machine (Wrap-Around RBM). Its setup is illustrated in Figure B.1. It has a few interesting characteristics:

- It can be trained on patterns which are not offset, but will still respond appropriately if the same patterns are input in an offset manner (i.e. training on one position generalizes to all other positions).
- The activation of the RBM is offset in exactly the same way as the input: one could superimpose a classifier with a similar wrap-around architecture, and not only classify patterns, but also identify their position.
- The system recognizes patterns with wrap-around. Hinton (1987) experimented with this before, but for the rest it seems that wrap-around pattern recognition has received little attention, or success (although practical applications are also limited).

The principle is quite simple: instead of training a normal RBM on only one position of the input pattern, for a single sample it is trained at all possible offsets, and the weights are effectively updated with the average gradient for all these offsets. In the implementation, this means that the input vector is extended such that the first values are repeated (as illustrated by the grey visible units in the figure), and that the RBM activations are calculated using a convolution operator. For the down-phase a similar mechanism is used. The exact details will not be described here.

So far, experimentation has only been done on (very) short input vectors with length 4, where two patterns where input: 1010 and 1100 and all their offset variants. Applying sparsity (Goh et al. 2010) resulted in quite promising results, where RBM activations could indeed be linked to which pattern was input and at which offset. In some cases the learned features in fact exactly resembled the input patterns, with two high values and two low values organized in the same sequence as the corresponding input vector to which it would respond.

Unfortunately, when longer input vectors were used or when sparsity was not implemented, results so far were not very successful. This does not have to mean that the approach does not work, it can also be that simply the right hyper-parameters were not found (RBM's are very sensitive to their configuration parameters, and the approach has not been thoroughly tested). Therefore, it might be worthwhile to experiment with this set-up further, and perhaps it could be applied for octave-compressed representations of input score (or perhaps even spectral information for audio) to learn about patterns in harmony.

Bibliography

Arcos & De Mántaras (2001), 'An interactive case-based reasoning approach for generating expressive music', *Applied Intelligence* **14**(1), 115–129.

Askenfelt (1986), 'Measurement of bow motion and bow force in violin playing', *The Journal of the Acoustical Society of America* **80**, 1007.

Baldi & Hornik (1989), 'Neural networks and principal component analysis: Learning from examples without local minima', *Neural networks* **2**(1), 53–58.

Bengio et al. (2006), Neural probabilistic language models, *in* 'Innovations in Machine Learning', Springer, pp. 137–186.

Bengio et al. (2009), Curriculum learning, *in* 'Proceedings of the 26th annual international conference on machine learning', ACM, pp. 41–48.

Bengtsson & Gabrielsson (1977), 'Rhythm research in uppsala', *Music, Room and Acoustics* **17**, 19–56.

Bergstra et al. (2010), Theano: a CPU and GPU math expression compiler, *in* 'Proceedings of the Python for Scientific Computing Conference (SciPy)'. Oral Presentation.

Bergstra et al. (2011), Algorithms for hyper-parameter optimization, *in* '25th Annual Conference on Neural Information Processing Systems (NIPS 2011)'.

Bielecki (2010), 'Genres', `http://en.chopin.nifc.pl/chopin/genre/search`. Consulted June 25, 2013.

Bisesi & Parncutt (2010), 'An accent-based approach to automatic rendering of piano performance: Preliminary auditory evaluation', *Archives of Acoustics* **36**(2), 283–296.

Boulanger-Lewandowski, Bengio & Vincent (2012), 'Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription', *Proceedings of the 29th International Conference on Machine Learning (ICML)* .

Bresin (1998), 'Artificial neural networks based models for automatic performance of musical scores', *Journal of New Music Research* **27**(3), 239–270.

Bresin, De Poli & Ghetta (1995), A fuzzy approach to performance rules, *in* 'Colloquio di Informatica Musicale-XI CIM', pp. 163–166.

Carreira-Perpinan & Hinton (2005), On contrastive divergence learning, *in* 'Artificial Intelligence and Statistics', Vol. 2005, p. 17.

Clarke (1988), 'Generative principles in music performance.'.

De Poli, Irone & Vidolin (1990), 'Music score interpretation using a multilevel knowledge base', *Journal of New Music Research* **19**(2-3), 137–146.

Desain & Honing (1992), *Music, Mind and Machine. Studies in Computer Music, Music Cognition and Artificial Intelligence*, Amsterdam: Thesis Publishers.

Desain & Honing (1994), 'Does expressive timing in music performance scale proportionally with tempo?', *Psychological Research* **56**(4), 285–292.

Elman (1993), 'Learning and development in neural networks: The importance of starting small', *Cognition* **48**(1), 71–99.

Erhan, Bengio, Courville & Vincent (2009), Visualizing higher-layer features of a deep network, Technical report, Technical report, University of Montreal.

Flossmann et al. (2010), 'The magaloff project: An interim report', *Journal of New Music Research* **39**(4), 363–377.

Foxley (2001), 'Nottingham Music Database', `http://www.chezfred.org.uk/University/music/database.htm`. Consulted October 24, 2013.

Friberg et al. (1991), 'Performance rules for computer-controlled contemporary keyboard music', *Computer Music Journal* **15**(2), 49–55.

Gabrielsson & Juslin (1996), 'Emotional expression in music performance: Between the performer's intention and the listener's experience', *Psychology of music* **24**(1), 68–91.

Goh et al. (2010), Biasing restricted boltzmann machines to manipulate latent selectivity and sparsity, *in* 'NIPS workshop on deep learning and unsupervised feature learning'.

Grachten & Krebs (2014), 'An assessment of learned score features for modeling expressive dynamics in music', *Transactions on multimedia: Special issue on music data mining* . Article in press.

Grachten & Widmer (2011), Explaining musical expression as a mixture of basis functions, *in* 'Proceedings of the 8th Sound and Music Computing Conference (SMC 2011)'.

Grachten & Widmer (2012), 'Linear basis models for prediction and analysis of musical expression', *Journal of New Music Research* **41**(4), 311–322.

Greentree (2011), 'JSBChorales.net: Bach Chorales', `http://www.jsbchorales.net`. Consulted October 24, 2013.

Grosjean, Grosjean & Lane (1979), 'The patterns of silence: Performance structures in sentence production', *Cognitive Psychology* **11**(1), 58–81.

Hinton (1987), Learning translation invariant recognition in a massively parallel networks, *in* 'PARLE Parallel Architectures and Languages Europe', Springer, pp. 1–13.

Hinton (2010), 'A practical guide to training restricted boltzmann machines', *Momentum* **9**, 1.

Hinton, Osindero & Teh (2006), 'A fast learning algorithm for deep belief nets', *Neural computation* **18**(7), 1527–1554.

Hinton & Salakhutdinov (2006), 'Reducing the dimensionality of data with neural networks', *Science* **313**(5786), 504–507.

Hinton & Sejnowski (1986), 'Learning and relearning in boltzmann machines', *MIT Press, Cambridge, Mass* **1**, 282–317.

Hochreiter & Schmidhuber (1997), 'Long short-term memory', *Neural computation* **9**(8), 1735–1780.

Honing & de Haas (2008), 'Swing once more: Relating timing and tempo in expert jazz drumming', *Music perception* **25**(5), 471–476.

Ishikawa (2000), Extraction of musical performance rules using a modified algorithm of multiple regression analysis, *in* 'Proc. of ICMC, 2000', ICMA.

Jaffer (2011), 'Voluntocracy', `http://www.voluntocracy.org`. Consulted October 24, 2013.

Juslin & Sloboda (2001), *Music and emotion: Theory and research.*, Oxford University Press.

Juslin & Vastfjall (2008), 'Emotional responses to music: The need to consider underlying mechanisms', *Behavioral and brain sciences* **31**(5), 559.

Katayose & Inokuchi (1993), 'Learning performance rules in a music interpretation system', *Computers and the Humanities* **27**(1), 31–40.

Krueger (2013), 'Classical Piano Midi Page', `http://www.piano-midi.de`. Consulted October 24, 2013.

Kveraga, Ghuman & Bar (2007), 'Top-down predictions in the cognitive brain', *Brain and cognition* **65**(2), 145–168.

Langner & Goebl (2003), 'Visualizing expressive performance in tempo-loudness space', *Computer Music Journal* **27**(4), 69–83.

Large, Palmer & Pollack (1995), 'Reduced memory representations for music', *Cognitive Science* **19**(1), 53–93.

LeCun et al (1995), 'Learning algorithms for classification: A comparison on handwritten digit recognition', *Neural networks: the statistical mechanics perspective* **261**, 276.

Lee, Ekanadham & Ng (2007), Sparse deep belief net model for visual area v2, *in* 'Advances in neural information processing systems', pp. 873–880.

Lerdahl (2001), *Tonal pitch space*, Oxford University Press.

Lerdahl & Jackendoff (1983), *A generative theory of tonal music*, The MIT Press.

Métin & Frost (1989), 'Visual responses of neurons in somatosensory cortex of hamsters with experimentally induced retinal projections to somatosensory thalamus', *Proceedings of the National Academy of Sciences* **86**(1), 357–361.

MuseScore BVBA (2013), 'MuseScore', `http://www.musescore.org`. Consulted October 24, 2013.

Narmour (1977), *Beyond Schenkerism: The need for alternatives in music analysis*, University of Chicago Press Chicago.

Nettheim (2013), 'The reconstitution of historical piano recordings: Vladimir de Pachmann plays Chopin's Nocturne in E minor', *Music Performance Research* **6**, 97–125.

Norouzi, Ranjbar & Mori (2009), Stacks of convolutional restricted boltzmann machines for shift-invariant feature learning, *in* 'Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on', IEEE, pp. 2735–2742.

Palmer (1989), 'Mapping musical thought to musical performance.', *Journal of experimental psychology: human perception and performance* **15**(2), 331.

Palmer (1997), 'Music performance', *Annual review of psychology* **48**(1), 115–138.

Parncutt (2003), 'Accents and expression in piano performance', *Perspektiven und Methoden einer Systemischen Musikwissenschaft* pp. 163–185.

Patel (2003), 'Language, music, syntax and the brain', *Nature neuroscience* **6**(7), 674–681.

Piston, DeVoto & Jannery (1978), *Harmony*, Norton New York.

Repp (1990), 'Patterns of expressive timing in performances of a beethoven minuet by nineteen famous pianists', *The Journal of the Acoustical Society of America* **88**, 622.

Repp (1992), 'Diversity and commonality in music performance: An analysis of timing microstructure in schumannÕs ÔÔträumereiÕÕ', *The Journal of the Acoustical Society of America* **92**, 2546.

Repp (1994), 'On determining the basic tempo of an expressive music performance', *Psychology of Music* **22**, 157–157.

Repp (1995), 'Quantitative effects of global tempo on expressive timing in music performance: Some perceptual evidence', *Music Perception* pp. 39–57.

Rifai et al. (2011), Contractive auto-encoders: Explicit invariance during feature extraction, *in* 'Proceedings of the 28th International Conference on Machine Learning (ICML-11)', pp. 833–840.

Roe et al. (1992), 'Visual projections routed to the auditory pathway in ferrets: receptive fields of visual neurons in primary auditory cortex', *The Journal of neuroscience* **12**(9), 3651–3664.

Sadie (1980), 'The new grove dictionary of music and musicians'.

Schenker (1969), 'Five graphic music analyses'.

Seashore (1938), *The psychology of music*, Courier Dover Publications.

Sloboda (1983), 'The communication of musical metre in piano performance', *The quarterly journal of experimental psychology* **35**(2), 377–396.

Steinbeis, Koelsch & Sloboda (2006), 'The role of harmonic expectancy violations in musical emotions: Evidence from subjective, physiological, and neural responses', *Journal of Cognitive Neuroscience* **18**(8), 1380–1393.

Sundberg, Askenfelt & Frydén (1983), 'Musical performance: A synthesis-by-rule approach', *Computer Music Journal* **7**(1), 37–43.

Sutskever, Martens & Hinton (2011), Generating text with recurrent neural networks, *in* 'Proceedings of the 2011 International Conference on Machine Learning (ICML-2011)'.

Suzuki, Tokunaga & Tanaka (1999), A case based approach to the generation of musical expression, *in* 'IJCAI', Citeseer, pp. 642–648.

The Mutopia Project (2013), 'The Mutopia Project', `http://www.mutopiaproject.org`. Consulted October 24, 2013.

Tibshirani (1996), 'Regression shrinkage and selection via the lasso', *Journal of the Royal Statistical Society. Series B (Methodological)* pp. 267–288.

Todd (1985), 'A model of expressive timing in tonal music', *Music Perception* **3**, 33–58.

Todd (1989), 'Towards a cognitive theory of expression: The performance and perception of rubato', *Contemporary Music Review* **4**(1), 405–416.

Tran (2009), 'Music analysis: ballades', `http://www.ourchopin.com/analysis/ballade.html`. Consulted June 25, 2013.

Van Essen et al. (1992), 'Information processing in the primate visual system: an integrated systems perspective', *Science* **255**(5043), 419–423.

Vincent et al. (2008), Extracting and composing robust features with denoising autoencoders, *in* 'Proceedings of the 25th international conference on Machine learning', ACM, pp. 1096–1103.

Widmer (1996), 'Learning expressive performance: The structure-level approach', *Journal of New Music Research* **25**(2), 179–205.

Widmer (2000), Large-scale induction of expressive performance rules: First quantitative results, *in* 'In Proceedings of the International Computer Music Conference (ICMC'2000). San Francisco, CA: International Computer Music Association', Citeseer.

Widmer (2003), 'Discovering simple rules in complex data: A meta-learning algorithm and some surprising musical discoveries', *Artificial Intelligence* **146**(2), 129–148.

Widmer et al. (2003), 'In search of the horowitz factor', *AI Magazine* **24**(3), 111–130.