Neural Machines for Music Recognition

Leo Pape
Utrecht University

March 26, 2006

Supervisors:
Dr. Marco A. Wiering
Dr. Frans Wiering
Dr. Vincent van Oostrom

Table of contents

Table of contents **i**

Abstract **iii**

Acknowledgements **v**

1 Introduction **1**

1.1 Pattern recognition . . . 1
1.2 Time series data . . . 4
1.3 Research question . . . 5
1.4 Relevance to artificial intelligence . . . 6
1.5 Thesis outline . . . 7

2 Neural networks **9**

2.1 Biological neural networks . . . 9
2.2 Introduction to artificial neural networks . . . 11
2.3 Feedforward neural networks . . . 13
2.4 Recurrent neural networks . . . 16

3 The liquid state machine **19**

3.1 Problems with recurrent neural networks . . . 19
3.2 The liquid state machine . . . 21
3.3 Neural circuitry as liquid . . . 23
3.4 Readout mechanisms . . . 24

4 The Helmholtz machine **27**

4.1 Introduction to pattern theory . . . 27
4.2 Processes, patterns and models . . . 29
4.3 Basic principles of the Helmholtz machine . . . 30
4.4 The wake-sleep algorithm . . . 33

5 Music **37**

5.1 Introduction to terminology . . . 37
5.2 Music perception . . . 38
5.3 Music classification . . . 40

6 Methods and implementation **43**

6.1 Music classification as learning task . . . 43
6.2 Setup . . . 44
6.3 MIDI to time series data . . . 46
6.4 Liquid filter and liquid states . . . 47
6.5 Readout mechanisms . . . 49

7 Experiments and results **53**
7.1 Overview 53
7.2 Readout mechanisms 54
7.3 Liquid optimization 56
7.4 Improving performance 60

8 Conclusion **63**
8.1 Discussion 63
8.2 Further research 65

Bibliography **ix**

Abstract

Since the early days of neural networks research, the focus has largely been aimed at forward processing of information. A recently developed theory called pattern theory, takes a different approach to solving pattern recognition problems. Rather than extracting features from input patterns, pattern theory aims at studying the interaction between the representation of input patterns, and the reconstruction of the input patterns from internal representations. An implementation of the principles of pattern theory in the form of a neural network is the Helmholtz machine. The Helmholtz machine, just as several other types of neural networks is very limited in the sense that it cannot learn dependencies between successive input patterns in time series data. Adjusting the architecture and learning algorithm of a neural network such that it can learn temporal dependencies often introduces a lot of problems. The liquid state machine presents a solution to some of these problems by applying a liquid filter that does not need to be trained, but can still be used as a source of information on temporal dependencies in time series data. A readout mechanism, such as the Helmholtz machine can be trained to extract information about temporal dependencies in the input data from snapshots of the liquid's state.

In this thesis the feasibility of the liquid state machine with a Helmholtz machine as readout mechanism is investigated, and compared with a feedforward neural network as readout mechanism. The learning task is the classification of classical music for keyboard instruments written by Bach and Beethoven. The liquid state machine and the Helmholtz machine are not just mere theoretical inventions; their inventors were also inspired by recent insights in the low-level structure of the human brain. Therefore, we will investigate some basic properties of biological neural networks as well as the representation of audio and music in the human brain in order to create a biologically plausible representation of the learning task.

Acknowledgements

First and foremost, I would like to thank Marco Wiering for being my first supervisor. I greatly appreciate all his time spent in discussing our ideas, reading the many draft versions of this thesis, and pointing me at methods and possibilities for further analysis. I would also like to express my thanks to my second supervisor, Frans Wiering, for reading and commenting this thesis. Thanks for being a critical reader and for the many suggestions to make this thesis more readable. Furthermore, I like to thank my third supervisor, Vincent van Oostrom, for reading this thesis, and listening to my stories about pattern theory and neural networks. Finally, I like to thank Michelle for letting me use her computer to run the experiments, when mine suffered from memory failure.

Whoever, in the pursuit of science, seeks after immediate practical utility, may generally rest assured that he will seek in vain.

HERMANN LUDWIG FERDINAND VON HELMHOLTZ (1821–1894)

Chapter 1

Introduction

1.1 Pattern recognition

The outside world is perceived by human beings through the senses. In the outside world all sorts of processes are going on that determine how the world is shaped. These processes generate patterns that are picked up by the senses to provide us with information about the state of the world. For instance, vibrations in the air are perceived as sound by the auditory system, and electromagnetic waves of certain wavelengths as color by the visual system. Observations by the senses are however not identical to patterns in the outside world. Brains do not process sound vibrations or electromagnetic waves, but operate by transmitting electrochemical signals between individual brain cells. Therefore the patterns in the outside world are transformed into a neural encoding in the peripheral zones of the sensory organs. Observations from the outside world represented as patterns of electrochemical signals can subsequently be processed and interpreted by our brain, which makes it possible to learn, make decisions, plan actions, and execute them, to manipulate the outside world as we see fit. Fortunately, the rules that determine how the outside world is shaped and how patterns are formed are regular. Although there are many different rules and processes that govern the behavior of objects in the outside world, it is still the case that the generated patterns have certain properties in common. This allows for interpreting observations by dividing them into separate classes which have certain properties in common. For example the class of liquids has the property that one cannot normally walk on it, and the class of solids that one can (except if it is a thin layer of ice). These properties or *features* can either be learned by extracting them from direct observations, or by interpreting and combining other observations. The process of extracting features is very important in perception. Incoming signals from the sensory system are not perceived as the 'raw' data of which they consist, but are immediately transformed by the brain into features, which can on their turn be transformed to even higher-level concepts. The processes in the human brain that are responsible for classifying input patterns, and the methods by which the brain adapts itself to improve this ability have been studied for several decades.

Since the age of digital computers the working of the brain has been compared to that of computers, and computers have been programmed to model certain aspects of the brain. Methods in which computers are used to mimic the capacity of the human brain to learn from observations, are called machine learning methods. Patterns generated in the outside world by stochastic processes share some statistical regularities. The task of a machine learning system is to extract those regularities and represent them as features. This can be realized by providing the system with examples of input patterns, and train the system by adjusting the parameters such that it learns to extract the features from the observed data. There are basically two possible applications of this method. The first deals with learning tasks in which the input patterns belong to certain classes, and the correct label of the class for each input pattern is already known. The input patterns are provided to the machine learning system together with their corresponding labels. The task of the system is to assign the correct label to each input pattern, relying on the extracted features. When the training process is finished, the system can be used for classification of novel patterns, for which the corresponding class is not known.

This process is called *supervised learning.* The second application is *unsupervised learning.* Unsupervised learning involves finding regularities in input patterns, while there is no a priori information about the class to which each input pattern belongs. Unsupervised learning can be used for extracting features from input patterns, without the need for assigning labels to the classes found during the training process. This method is sometimes used for preprocessing the input data, where the extracted features can be applied in other learning methods.

Nowadays there are a vast number of pattern recognition problems that can be successfully solved by machine learning algorithms, such as the classification of emails into spam or non-spam categories, recognition of handwritten postal codes on envelopes, or recognition of images of human faces. As in most areas machines do not perform as well as humans, machine learning tasks are often limited to learning examples from a specific domain. The problem that is to be solved by a machine is called the *learning task.* There exists a number of different machine learning methods, most of them based on statistical methods but sometimes also inspired by the actual biological structure of the brain, that can be applied on learning tasks. Some learning tasks require machine learning methods that are specifically designed for the task, while other tasks can be solved by a number of different learning algorithms. A subclass of machine learning methods that has been studied almost as long as there are digital computers, is the class of artificial *neural networks.* Artificial neural networks are somehow based on the low-level structure and properties of the human brain, but are in contrast to their biological counterparts implemented on electronic circuits or in computer programs. These networks consist of simple computation units, the equivalent of brain cells, that send signals to each other through connections between them. A large number of connected units each performing relatively simple computations can together solve complex tasks that are comparable to problems the human brain can deal with.

The structure of artificial neural networks is often layered, not only to simplify the mathematical properties or implementation in a computer program, but also to build a hierarchical model in which each subsequent layer extracts more abstract properties from the previous layer. Feature extraction is the most prominent method that is used by neural networks and several other types of machine learning methods, to assign labels to input patterns in a classification task. Compared to the human brain, this seems plausible because we perceive the outside world not as the 'raw' sensory patterns, but sensory information is immediately and effortlessly combined and translated to the more significant *meaning* of the input data. Most artificial neural networks do nothing more than the feedforward processing of information from layer to layer into higher-level features and concepts. In the field of psychology however, many experiments have shown that what we perceive is neither the 'raw' sensory data, nor the combination of features that is extracted from it, but a *reconstruction* of what it is like. For example, in an experiment from the field of cognitive psychology known as the 'Mooney faces', different images

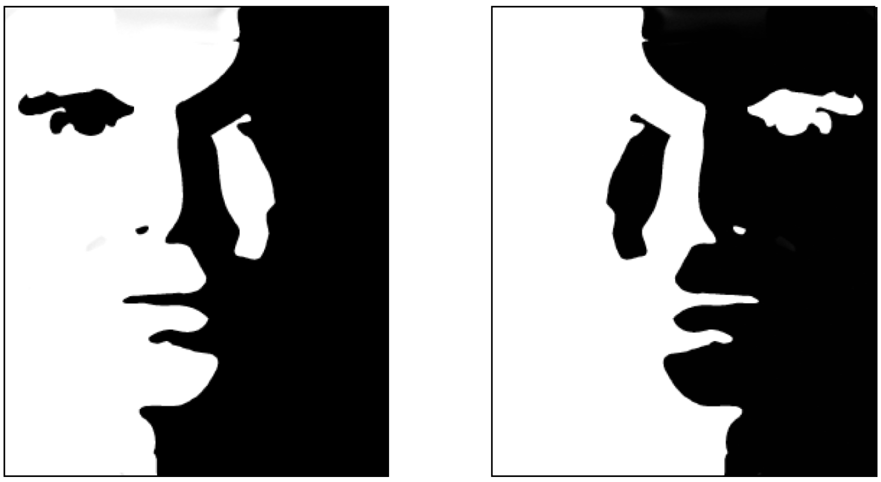


Figure 1.1: The figure left is a 'Mooney face' and the figure right is the mirrored image with black and white inverted

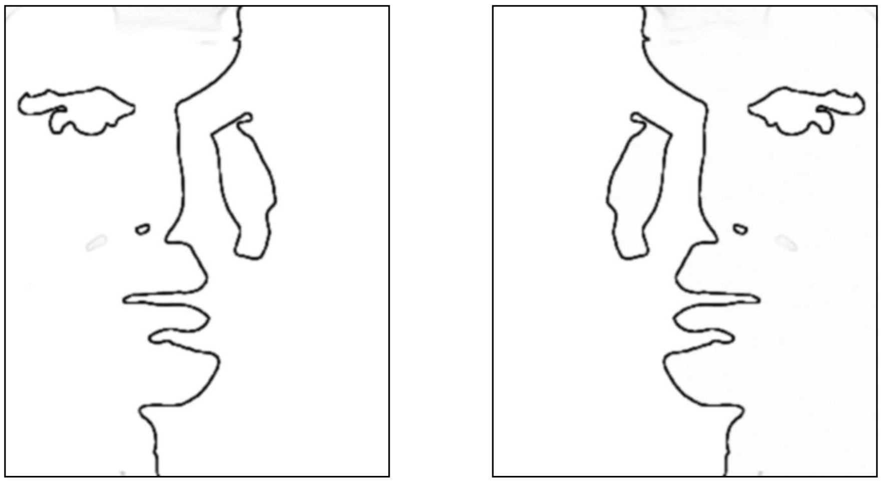


Figure 1.2: An edge detection algorithm applied to both images from figure 1.1 reveals no difference

of faces are depicted as in figure 1.1 (example adapted from Mumford and Desolneux 2005). In the left image, there is a strong light which illuminates part of the face to saturation and leaves the rest dark. In the right (mirrored) image the effect is reversed. An example of a basic feature extraction algorithm applied on both images in figure 1.1 is depicted in figure 1.2. This method of feature extraction reveals no difference between the two images, except for the fact that they are mirrored, and the results do not look like a face at all. The fact that in figure 1.1 the left image is easily recognized as a face while the right is not, implies that there must be information in the brain about the appearance of faces that is used to fill in the missing parts of the face, thus constructing the perception of a face. If the contrast is reversed, the information the brain has about the appearance of faces cannot be related to the image at all, and no face is recognized. Just extracting features does not seem to be the way in which the task of recognizing a face can be solved. This example as well as a number of other results from the field of cognitive psychology implies that feature extraction by feedforward processing of information is not the only process that takes place in the brain; the reconstruction of input patterns might also play a significant role.

In pattern theory (Grenander, 1981; Mumford, 1994) this idea is investigated in more depth, from a computational as well as a biological point of view, to create a better explanation for the complex processes that are involved in pattern recognition. The theory introduces some general properties that models should satisfy, and states a number of hypothesis about the structure and processes in the brain, some of which are more or less supported by actual findings. Although experiments from the field of psychology that were performed several decades ago already suggested that pattern reconstruction is at least as important as feature extraction, pattern theory in its present form has existed for only ten years. It is however interesting that certain problems and discrepancies between biological and artificial systems that have haunted the scientists in the field of neural networks for a long time can be solved with this new theory. The basic idea in pattern theory is that a pattern recognition system not only learns to extract features from input patterns, but uses these features to reconstruct the input patterns in order to recognize them. Apart from the solipsist's point of view, patterns that enter a system from the outside world are caused by certain regular processes that determine how the outside world is shaped. Learning the way in which processes in the outside world generate those patterns is the most economic solution to the task of recognizing input patterns. To reconstruct input patterns from higher level features, pattern theory adds a generative model to already existing recognition models in the field of neural networks, whose task is to learn to represent the actual processes that caused the input patterns to occur. The idea used in mere feedforward processing methods that parts of the learning task could easily be solved by preprocessing the input patterns is abandoned in pattern theory because it often adds more complexity to the task of reconstructing the input patterns.

In pattern theory, the brain or some other neural network is seen as a statistical inference engine that learns to infer a probability distribution over the set of causes in the outside world that generated the input patterns. This probability distribution can be used for recognizing novel input patterns. The idea that the perceptual system uses some kind of generative model to provide a probability distribution over the probable causes of the input was already stated by a 19th century physicist and mathematician, called Hermann von Helmholtz. More than a century after he died, the math he invented to describe physical systems in terms of statistics was beautifully combined with the basic ideas from pattern theory in the form of a neural network named after him; the Helmholtz machine. Helmholtz machines have already successfully been applied to pattern recognition tasks such as extracting depth from stereo images, handwritten character recognition and text classification (Dayan et al., 1995; Hinton et al., 1995; Chang and Zhang, 2001).

1.2 Time series data

Machine learning systems are almost always implemented on digital computers. This imposes certain restrictions on the process of pattern recognition, and determines some specific properties of machine learning systems with respect to the human brain. Just as the brain, computers cannot handle information from the outside world in its basic form. A computer can only process numbers, not colors or sounds. Therefore the input data in machine learning systems have to be translated into a form that can be processed by a digital computer. The actual input data to a machine learning system are not in the original form that was determined by the processes that created them, but are representations of those data in a numeric form. A major difference with the human brain is that all processes in a digital computer are governed by a central clock, and all operations are performed in discrete time steps. This implies that patterns can only be provided at discrete time steps. The representation of a pattern in the world outside the machine learning system in a numerical form that is provided to the system at a discrete time step will from now on be referred to as an *input pattern*.

Because of the discrete nature of programs implemented on digital computers training examples in most learning tasks consist of case-by-case 'snapshots' of a situation. This is not a problem in case each separate input pattern contains enough information for correct classification. However, especially when dealing with phenomena or behavior expressed by human beings, the structure of the related patterns shows temporal dependencies between the different input patterns. The temporal dependencies are now crucial for correct classification, and a learning algorithm depends on the extraction of information from those temporal dependencies. In speech recognition for example, it isn't of much use to know that the current input pattern is classified as an '*n*', when one wants to recognize the word '*neuron*', if one cannot relate the '*n*' to the other symbols. Learning the temporal structure inherent in such patterns involves learning the dependencies in the observed time series of patterns. When the temporal dependencies between individual input patterns span over longer time periods, it is very difficult to create a model and a corresponding learning algorithm that can extract information from these long-term temporal dependencies. The problem is that supervised learning methods in the field of neural networks, should be able to store information for the period of time that is long enough to contain the temporal dependencies. It is however not a priori known how long this period is, and where it starts or ends. Learning to extract information on patterns the learning method has seen many time steps ago complicates the learning task in such a way that it rarely the case that reasonable results can be obtained (Hochreiter, 1998).

A solution to this problem was proposed in the form of the liquid state machine (Maass et al., 2002). The liquid state machine uses a static temporal filter that extracts temporal dependencies from successively presented input patterns. This filter is static in the sense that it is not necessary to train it for a specific learning task. The output of the filter is presented to a readout mechanism, for instance a neural network, that does not need to implement any temporal dependencies between successive input patterns because that information was already extracted by the liquid filter. How this works can be explained with an analogy to a real liquid. The time series of input patterns are injected in a liquid by disturbing the resting state of the liquid. These disturbances propagate through the liquid and create specific resonating patterns over time, just as a pebble that is thrown into the water creates ripples in the surface of the water. The state of the liquid at a certain time contains information on previous inputs, just as the waves in water contain information on the time and place that a pebble was dropped. In a classification task, this information can be presented at certain time steps to a readout mechanism that learns to map the state of the liquid to a certain class. A liquid filter, whether it is a real liquid or some other dynamical system can be used for a whole range of different tasks as it does not need to be adapted or trained for a specific task.

The activity in some areas in the brain can be interpreted as the equivalent of perturbations in a liquid. In these areas, incoming information creates certain patterns of activity that resonate through the network, and other areas can read out this information for further processing. Although the liquid state machine is based on actual brain structures, it is not just another interpretation of how the brain works, but the ideas also extend to models in machine learning tasks dealing with the classification of time series data. Real biological neural networks as well as artificial models can be seen as rather complex dynamical systems, which manifest more complex behavior than a simple liquid such as water. Moreover, as it is not necessary to train an artificial neural network that is used as a liquid filter for a specific task, the problems with the training of an artificial neural network on time series data disappear. This means that more complex recurrent neural networks can be applied as temporal filter, such as a spiking neural network (Maass et al., 2002; Vreeken, 2004).

1.3 Research question

Because the Helmholtz machine has the same problems when faced with time series data as other neural network architectures, it might be interesting to investigate certain solutions found in the field of neural networks, and apply them to the principles of pattern theory. A possible solution is to use the liquid filter of a liquid state machine as a temporal filter. The output of such a liquid filter can be presented to a Helmholtz machine for further processing. The question that needs to be answered is:

> *What are the capabilities of a Helmholtz machine as readout mechanism of a liquid state machine in comparison with other readout mechanisms?*

This question immediately gives rise to another question of a more theoretical nature: can the Helmholtz machine extract any information *at all* from a temporal liquid filter? From the previous it could be reasonably doubted whether this is possible, because the nature of the Helmholtz machine is very different from other learning mechanisms. Helmholtz machines are very good at learning pattern recognition from direct sensory input. Too much preprocessing on the sensory data just complicates the learning task for a Helmholtz machine, while that is exactly what a temporal filter does. However, given the possible solutions for dealing with time series data, the liquid state machine offers not only a solution that is easy to understand and implement, but is also based on interpretations of actual brain processes. Faced with the problem of time series data in pattern theory, Mumford proposes some sort of "temporal buffering" for which the neuronal mechanisms "have not yet been described" (Mumford, 1994, page 26). If he was right, there must be a way to apply pattern theory to data that show temporal dependencies between different input patterns. The only way to find out whether this can be done with a liquid filter as in the liquid state machine is to try.

Other interesting questions that arise from the main question are: what kind of liquid should be used, how should the information from the liquid be presented to the Helmholtz machine, what other readout mechanisms can be used and what learning algorithms for the readout mechanisms can be used? The answers to those questions also need to be found by running experiments and comparing and interpreting the results.

It is common practice in the field of machine learning that researchers test their newly designed algorithms on rather simplistic problems created by simple mathematical processes. These problems called *toy problems*, can give a better understanding of the studied algorithm, but don't necessarily say something about the performance of the algorithms on more realistic problems, such as the problems the human brain can solve. A more complex real-world problem tells us much more about the performance of an algorithm or structure of an artificial neural network, but is of course also more difficult to learn. Because there are generally speaking a lot of parameters that can be adjusted it can take quite some time to find the optimal parameter settings for a particular problem. Other problems of testing real-world data are that it is more difficult to obtain, and that it contains more noise that cannot be filtered out. However, despite all these problems, solving real-world problems tells us more about the real capacity of a machine learning mechanism, and above all, it speaks more to the imagination. Because both the Helmholtz machine and the liquid state machine are not mere mathematical inventions, but are based on interpretations of the actual physical structure and processes in the human brain, a learning task was selected that involves a real-world problem that can be solved by the human brain: the classification of music of Bach and Beethoven by composer. Music classification based on musical content between genres that share most harmonic properties is a task that cannot be solved by simply investigating individual snapshot of a musical piece, because the temporal dependencies between different notes and groups of notes play an important role. Classification of musical pieces is a typical problem that humans can easily learn to solve. Research has shown that there are certain structures in the human brain that are especially suited for processing music. Not only can these findings be used to build a model that mimics some of the actual brain processes that are involved in music recognition, but it also shows that music and brains are interwoven. Therefore music classification is a suitable learning task for this project.

1.4 Relevance to artificial intelligence

The human brain has been compared to many things in attempts to understand how it works. Theories about the complex interplay between the structure of the brain, and how it operates are often highly flavored by the prevailing technology of a particular time in history. In the days of Freud the technology of the steam engine and the science of hydraulics were becoming very popular. It is therefore not very surprising that Freud's theory about psychological phenomena contains references to the properties of the low-level structure of the brain in terms of 'building up pressure' that needs to be 'released', or the 'flow' of the libido. When the telephone came along, with it came the switchboard operator models of the brain. Such explanations of how the brain works in all its complexity could however not be used to model actual brain processes. Since the age of the digital computers, this has changed. Nobody would seriously think of building an intelligent steam engine, or sacrifice a telephone exchange center for scientific research, but computers are available for almost everybody nowadays, and they can be programmed to perform a large range of different tasks. Studying the brain is not limited anymore to the investigation of its actual structure while speculating about how it manages to perform intelligent tasks, but can also be extended to building computer models and compare their structure and the results of the tasks they can perform with the results of brain research.

Artificial intelligence is about simulating the intelligence that is found in processes and creatures in nature. This can either be done by building models that are related to actual brain structures, or by using other methods that are not necessarily related to the human brain but can still perform cognitive tasks at a comparable level of intelligence. The latter methods are often based on more formal properties of cognition, such as logic, formal language analysis and tight rules that combine the basic concepts of which they consist. Because the phenomena studied in artificial intelligence are often not strict, formal methods suffer from problems that arise from irregularities, ambiguities and fuzzy data. These properties of real-world problems are however effortlessly solved by the cognitive functions of the human brain. To solve the problems that arise from the irregular patterns in the real world, it might be a good idea to investigate how the brain manages to perform this. This is exactly the approach that was taken in pattern theory. The result is a theory in which irregular properties of real-world patterns naturally fit. Not only does this theory overcome the problems of more orthodox methods in the field of artificial intelligence, it is also possible to implement its basics in an artificial neural network. While it is possible to interpret the working of the brain at its low-level structure in terms of pattern theory, the theory has also certain problems. A major problem is that it cannot deal with time series data, for it requires a temporal buffer or short-term memory. Mumford recognized that a neural mechanism that could perform this task was not known at the time (Mumford, 1994). A solution to this problem for other types of neural networks was found in the form of the liquid state machine. Therefore it might be interesting to investigate if this solution can also be applied to a neural network that is based on pattern theory. As theories about intelligence come and go, new insights from brain research implemented in ideas, theories and models will hopefully contribute to a better understanding of intelligence and how it is accomplished by the brain.

1.5 Thesis outline

Since the brain, its structure and processes are the essence of what is studied and modeled in this thesis, we start with a short introduction to biological neural networks, and their building blocks: neurons. We continue with the description of artificial neural networks, and describe some similarities and differences with biological neural networks, the basic properties of the neuron, how neurons can be combined to form neural networks, and how neural networks can be trained. Next a description of a very simple, but elegant type of neural network, the feedforward neural network is presented. This type of network is used to explain the basics of training neural networks. In the last section of chapter 2 a description of neural networks that have been adapted for the task of learning time series data is given, followed by the problems that are introduced by their training algorithms.

Chapter 3 elaborates some more on the problems that arise from learning time series data with neural networks, and describes a solution to these problems in the form of the liquid state machine. The chapter continues with the characteristics of neural circuitry used as liquid filter, and the properties of readout mechanisms. At the end of the chapter the assumptions that are being made for neural networks that are solely based on feedforward processing, and the problems that arise from this perspective with respect to the liquid state machine are discussed.

Chapter 4 gives a more elaborate view on pattern theory. The properties of patterns, the relation with processes that create them and the models that can be build for recognition are investigated in more detail. In the last sections of this chapter a detailed description of the statistical physics that formed the basis of the mathematical background of the Helmholtz machine is given, together with the training algorithm that is derived from this mathematics.

Chapter 5 is about music and music perception. First the basic properties of sound and music are introduced, and some basic musical terms that will be used in the rest of the thesis will be explained. Next, the structures and processes that play an important role in music recognition in the brain are discussed, accompanied by the results of the research that led to this insight. It is of course not the first time that machine learning mechanisms are applied for the task of music classification. In the field of music classification the musical data are often preprocessed by all sorts of feature extraction methods. The results of these methods are then presented to machine learning methods, such as neural networks. Instead of relying on extensive feature extraction methods, the model used in this project represents the musical data in a more biologically plausible way. The last section of chapter 5 shows the difference between common practice in other music classification methods and the approach taken in this project.

Chapter 6 describes the setup of the experiments that were performed to investigate the questions in section 1.3. It gives a better insight in the implementation of the different machines and the choices that were made during their design. Furthermore some information on the classification task and how the music data were gathered is given. Many papers on the subject of machine learning are often vague about the specific details of the implementation and parameter settings. Because we believe that it is important to be able to reproduce the same results, an extensive description of the implementation of the neural networks as they are described in the previous chapters is given together with the methods that were used for training, testing and performance measuring. Also an overview of the adjustable parameters is given in tables, together with an explanation why certain values were chosen for the parameters, and why other values have to be found empirically.

In chapter 7 a description is given of the experiments that were performed and why they were performed in a certain form. The results of those experiments are shown graphically in this chapter.

Chapter 8 discusses the results found in the previous chapter, and the implications they have for the research questions. Some interesting possibilities and recommendations for further research are also given.

Chapter 2

Neural networks

2.1 Biological neural networks

Since the beginning of the computer era researchers have attempted to mimic the capacities of biological neural systems by modeling the low-level architecture of the brain. The human brain is composed of a large number of brain cells, called *neurons*, which are highly interconnected. The number of neurons in the average human brain is estimated around 100 billion (Kalat, 2001). Each neuron is a specialized cell that can receive, process and transmit electrochemical signals. A neuron, as depicted in figure 2.1(a), consists of a cell body, branching input structures called *dendrites*, and a branching output structure called an *axon*.

All parts of a neuron are covered by a membrane which in absence of any disturbances maintains an electrical polarization of the inside of the neuron with respect to the outside. This potential is called the *resting potential* of the neuron. One mechanism that maintains the resting potential of the neuron, is a pump that transports charged ions into and out of the cell. Because this pump transports more positively charged ions out of the cell than it transports into the cell, the net result is that the neuron has a negative resting potential. Furthermore, the membrane is covered with channels which can be closed or opened, allowing charged ions to pass through the membrane. At rest, these channels are closed, but when they open, positively charged ions will flow into the cell because of the difference in electrical potential between the inside of the cell and its environment.

The tips of the branches of the axon, or the presynaptic terminals, are connected to the dendrites of other neurons via *synapses*. The synapse, as depicted in figure 2.1(b), is a small junction in which chemicals from the tip of the branch of an axon are released, which then travel through the synaptic cleft and are received at the tip of the branch of a dendrite, called the postsynaptic terminal. The chemicals that are released in the synaptic cleft attach to special receptors at the postsynaptic terminal, which then inhibit or excite the membrane potential of the postsynaptic neuron. These chemicals or *neurotransmitters* can initiate many reactions or sequences of reactions in the postsynaptic neuron. The reactions that ultimately cause the ion channels in the postsynaptic neuron to open, allowing positively or negatively charged ions to flow into the cell, are responsible for the changes in the membrane potential. As the dendrites of a neuron have many synapses in which incoming chemicals affect the membrane potential of the neuron, the result in terms of membrane potential is a summation over all these disturbances. When the total membrane potential of the postsynaptic neuron reaches a certain positive threshold, a mechanism starts that opens the channels of the axon that allow the positively charged ions to freely flow into the cell. These channels open wider proportional to the positive potential of the axon's membrane, so increasing the membrane potential results in widening the ion channels, which causes the membrane potential to increase even more. Once the membrane potential is above the threshold the axon thus amplifies it to its maximum strength, enabling the potential to propagate very rapidly. This process always results in the same maximum activation value of the membrane potential, and takes very little time. The electrochemical pulses generated in this way are called *action potentials* or *spikes*, and the process of sending them is called *firing*. When such an electrochemical signal reaches the tips of

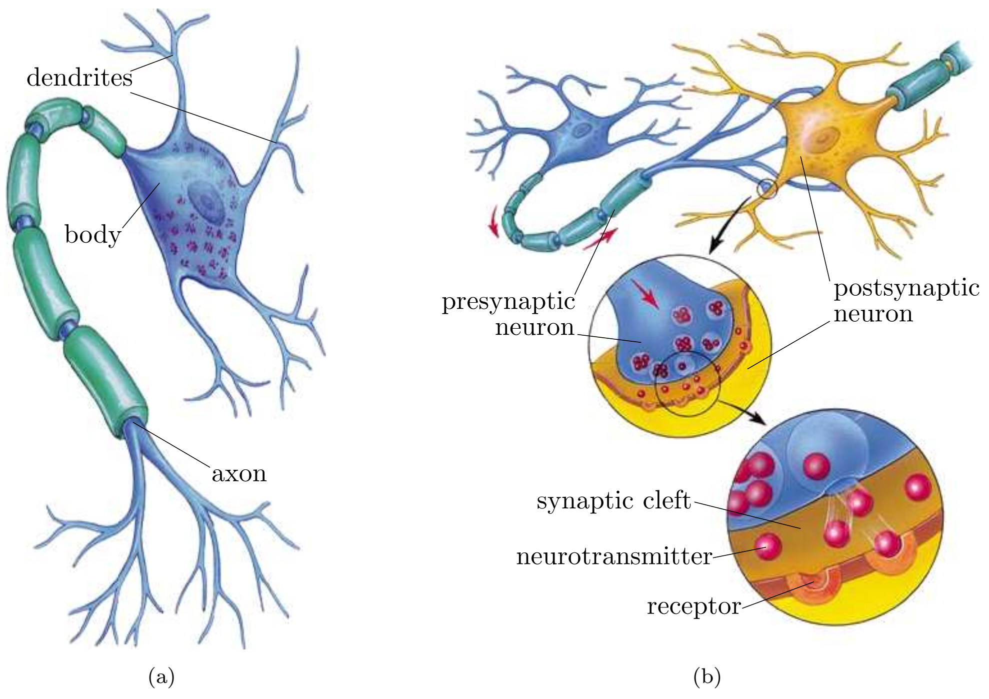


Figure 2.1: Overview of a biological neuron (a), and a synapse (b)

the branches of the axon, it triggers the release of chemicals which travel through the synaptic cleft to the dendrites of other neurons, which may in turn fire.

Each neuron synthesizes its own neurotransmitters, which costs a considerable amount of energy and time. Propagating an action potential through the synaptic cleft takes a certain amount of neurotransmitter molecules. Because the process of generating neurotransmitters runs at a certain speed and requires chemical building blocks and energy, the neuron can run out of neurotransmitters. After a period of rapid firing, the neuron can get 'tired' and cannot maintain its firing rate any longer. Furthermore, after each time the neuron fires, it enters a state in which it cannot fire again for a short period. Immediately after sending an action potential, the ion gates are firmly closed and the membrane cannot produce another action potential for a short time. This state is called the *absolute refractory period.* During the next state, some of the ion channels are closed, but not all, requiring a much stronger stimulus to produce another action potential. This stage is called the *relative refractory period.*

The total potentiation of the neuron depends on the spatial and temporal summation over the fluctuations of the membrane potential caused in the synapses on the branches of the dendrites. Due to the strongly branched architecture of the dendrites, the postsynaptic terminals are situated at different distances from the cell body. The amount of the postsynaptic potentiation that reaches the cell body is inversely proportional to the distance from the synapse to the cell body. Moreover, propagating the disturbances in the membrane potential caused in the tips of the branches of the dendrites costs an amount of time proportional to the distance to the cell body. This means that some synapses are more influential than others because of the spatial and temporal factors that influence the propagation of disturbances in the membrane potential. Furthermore, the strength of the signal received by a neuron through a synapse depends on the efficacy of the elements that make up the synapse, such as the amount of available neurotransmitter molecules, the amount, type and position of the receptor molecules, possible receptor blocker molecules and a whole lot of other factors. Together with the spatial and temporal aspects of the strength of the signal related to the location of the synapse, these factors determine the effect a synapse can have on the total potentiation of the membrane in the cell body, which is called the *strength* of the synapse.

The processes and variables described before are the basic mechanisms by which a neuron operates. A structure that consists of a very large number of these building blocks, such as the human brain, is capable of performing extremely complex and dynamic tasks. Moreover this structure has the ability to adapt itself to certain circumstances, and can somehow reorganize itself to improve its skills on different tasks. How this process, which is usually called *learning*, takes place, and by what mechanisms the brain can train itself, has been the subject of research for many years.

It was Hebb (1949) who first postulated the idea that learning consists principally of altering the strength of a connection by some growth process or metabolic change that takes place in one or both neurons that are interconnected. Later research has shown that such changes did indeed take place by a mechanism called long-term potentiation (Bliss and Lømo, 1973). Learning in a biological neural network that consists of interconnected neurons is achieved by strengthening the synapses between those neurons, if two or more presynaptic neurons repeatedly fire simultaneously and contribute to the firing of the postsynaptic neuron. The opposite change, long-term depression, can also occur, which is a prolonged decrease in response to slow or weak stimulation of the postsynaptic neuron. The mechanisms that are involved in these processes are very complex. Synapses can be strengthened or weakened, by altering the structure of postsynaptic receptors to become less or more sensitive to neurotransmitters, decreasing or increasing the number of certain types of postsynaptic receptors, removing or creating synapses between the two neurons, and various other processes.

In this fashion the brain manages to perform and learn extremely complex tasks using a very large number of neurons and connections. There is a great deal of complexity in biological neural networks that has not been discussed here, but these are the basic principles of the functioning and learning processes of the brain as they are understood nowadays. Most artificial neural networks can achieve some remarkable results with a model using not much more complex processes than these.

2.2 Introduction to artificial neural networks

The term artificial neural networks refers to a large class of different architectures with properties that are related to those of their biological counterpart. Artificial neural networks are mostly simulated by a computer program, or sometimes implemented on electronic circuits. Applications of these networks include not only attempts to simulate and model specific parts of the human brain to better understand the mechanisms by which it operates, but they are also used to solve certain types of problems, usually related to the type of problems that humans can deal with. Since the first attempts to mimic the brain by modeling its low-level structure, many researchers came up with their own ideas about the specific topology, building blocks and algorithms that could be used in neural networks, resulting in many architectures that belong to the large class of artificial neural networks.

All artificial neural networks have certain properties in common. Just as the human brain, they are built out of separate neurons, or *units*, that are connected to each other, and perform a certain operation on the input patterns to produce an output. Signals in an artificial neural network are of course not of electrochemical nature as in biological neural networks, but are usually represented as numbers. For example in some network architectures the output signal of a unit is 1 if it fires, and 0 otherwise.

A connection between two units has a certain weight related to it. When a signal is sent over the connection, it is modified by the weight of that connection. As learning in a biological sense is accomplished by adjusting the strength of the synapse, the equivalent of this mechanism in an artificial neural network is the modification of the weights of the connections.

Signals that enter a unit through the input connections are processed into a single number by a certain *summation function*, which is the equivalent of the total membrane potential of a biological neuron. Mathematically the input signals and corresponding weights of a certain unit i, can be represented as $(y_1, y_2, \ldots, y_n)$ and $(w_{i1}, w_{i2}, \ldots, w_{in})$, where y_j is the output signal of unit j, and w_{ij} is the weight of the connection between the output of unit j and the input of unit i. The simplest summation function is the dot product of the two vectors, which can be computed by multiplying each input signal with its corresponding weight, and adding all results:

$$\begin{aligned} net_i &= (w_{i1} \cdot y_1 + w_{i2} \cdot y_2 + \ldots + w_{in} \cdot y_n) \\ &= \sum_j w_{ij} y_j \end{aligned} \qquad (2.1)$$

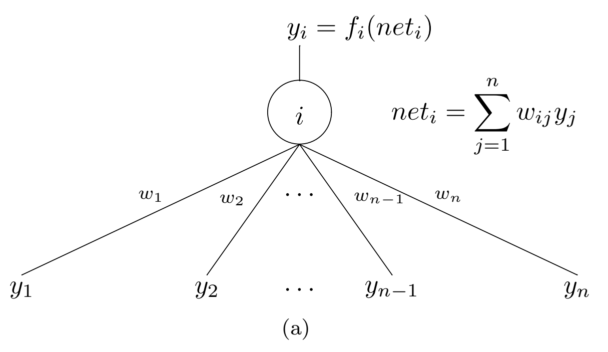
Of course other, more complex summation functions are possible which combine the input and weighting coefficients in different ways.

The result of the summation function is passed on to a *transfer*, or *squashing function* to compute the output of a unit. In the transfer function, the output of the summation function can be compared with a certain threshold, to determine the value of the output signal. Of course many other functions are possible, which usually are non-linear functions. Linear functions are very limited, because they produce an output that is proportional to the input. Minsky and Papert (1969) showed that neural networks using only linear functions cannot solve problems that are not linearly separable, and are therefore not so very useful for most problems. The output that is generated by the transfer function is sent through the output connections to the units that are connected to it. Just as in a biological neuron, an artificial neuron generates just one output, which can be sent to many other units, while it can receive signals from many other units. An example of an artificial neuron is given in figure 2.2(a).

Patterns enter a network by providing them to the input connections of particular units called *input units*. These input signals are collected by the summation function, and the output is computed by the transfer function. Next, the output of the transfer function is sent to the inputs of connected units. This process is repeated in each subsequent unit whose inputs are connected to the output of the sending units. In this way the signals travel through the network. An example of an artificial neural network is depicted in figure 2.2(b).

To solve a certain type of problem, a neural network usually has to be trained on a set of examples, called the training set. There is one important feature of the training set that determines the architecture that can be used to solve the problem. In the case that for each example in the training set the target value is given, a supervised architecture can be applied. In case there exist no target values of the input patterns that should be learned, only unsupervised learning methods are feasible.

Supervised neural networks try to learn to produce the correct output, given the input data. Usually, the process of learning starts by randomly initializing the weights of the network. Next, the examples are provided to the network, which generate certain activities in the units. The activity of some or all of the units, called the *output units*, is compared to the desired output by computing the difference between actual and the desired output. This difference or *error* is then transformed by an *error function* matching a particular network architecture. Using the



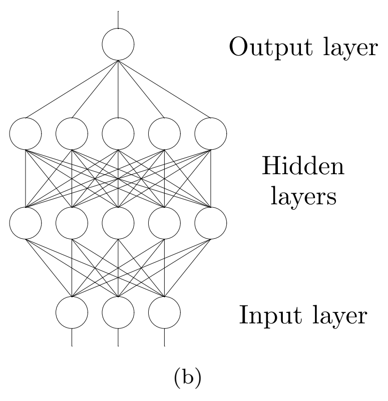


Figure 2.2: Example of an artificial neuron (a), and an artificial neural network (b)

results of the error function, the weight changes that are required to better match the targets given the corresponding input pattern and outputs are computed.

Unsupervised learning methods somehow learn to organize the input data themselves, according to a certain similarity metric. Just as in supervised neural networks, learning in unsupervised networks is often started by randomly initializing the weights. Input data are provided to the input units, and are propagated through the network. In contrast to supervised learning, where the error function is based on the difference between the desired and the actual output, the error function in unsupervised learning is based on certain characteristics of the input data.

Each network architecture uses its own learning function to compute the desired weight change for a connection. The weight changes can be applied after the presentation of each input pattern, or they can be accumulated over the presentation of several examples. Usually it takes the presentation of a large number of examples until a network produces sensible results. When no further learning is required because the results produced by the network are accurate enough, the weights are frozen and the network can be used to produce outputs for novel input patterns.

Learning a neural network involves adjusting the weights, using a *training algorithm*. A training algorithm consists of a method to compute the error in case of supervised learning, and a rule for determining the desired weight changes. The first learning rule was introduced by Hebb, and is based on learning in biological neural networks. In terms of artificial neural networks this rule means that if a unit receives an input signal from another unit and both are highly active, the weight between the units should be strengthened (Hebb, 1949). Since the introduction of this principle, many ideas have emerged that can be used to improve the performance of artificial neural networks. Some examples are; the introduction of a learning rate which specifies the magnitude of the strengthening or weakening of the weights, or the gradient descent method which transforms the error in each unit by the derivative of the transfer function of that unit.

2.3 Feedforward neural networks

One of the most widely known and commonly used artificial neural network architectures is the *feedforward neural network*, also known as the *backpropagation network*, or the *multilayer perceptron* (Rumelhart et al., 1986). Feedforward neural networks using non-linear transfer functions are capable of solving an extensive range of problems. In such a network, the units are organized within separate layers. In a fully connected multilayer feedforward network, each unit in one layer is connected to every unit in the layer below. An input pattern is provided to the lowest layer, and the signals travel through the network from layer to layer, to the top layer. The activities in the top layer, or output layer, are then compared to the desired targets to compute the error. The network is trained by the backpropagation algorithm, which computes the weight update for each connection. An example of the structure of a feedforward network is depicted in figure 2.2(b).

The specification of a feedforward neural network can best be given in mathematical terms. Let $\mathcal{N}$ be a neural network. The external inputs to the network and the outputs of the units are denoted by y. H is the set containing indices i, such that y_i is the output of a unit, I is the set containing indices i, for which y_i is an external input to the network, and K is the set containing indices i, for which y_i is the output of a unit in the hidden layers of the network. Let $\vec{w}$ be the weight vector with a unique value w_{ij}, for each weight between the input of unit i and the output of unit j of every connected pair of units in the network. It is possible that at some point all inputs to the network are 0, whereas the output should be a number other than 0. To enable this kind of behavior the units include a bias term, so each unit that is not an input unit is connected to a single bias unit, which always has a value of 1. Let

$$net_i = \sum_{j \in K \cup I} w_{ij} y_j \tag{2.2}$$

be the net input to the i^{th} unit for $i \in H$. The output of the unit is:

$$y_i = f_i(net_i), \tag{2.3}$$

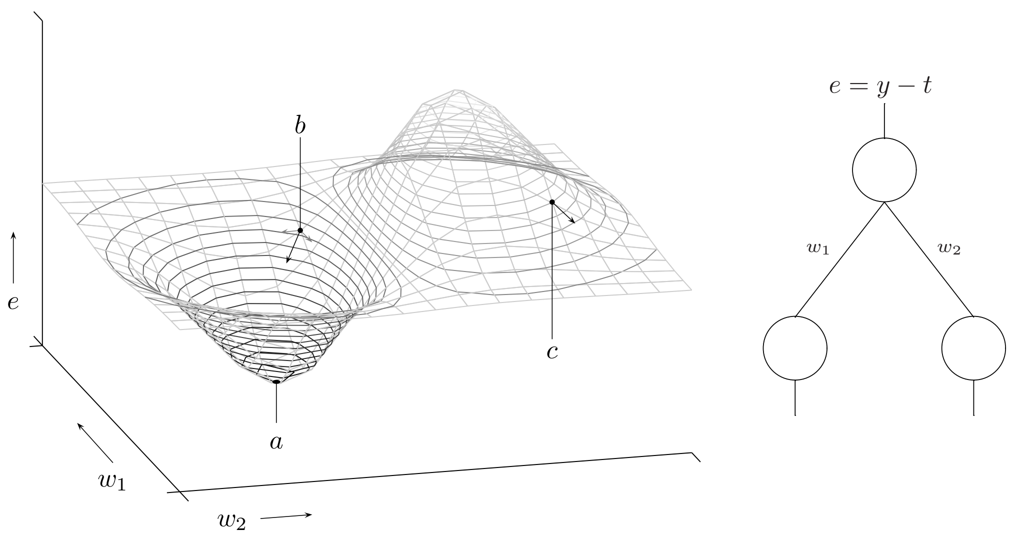


Figure 2.3: Weight space with respect to the error

where f_i is the unit's squashing function. Note that f_i can be any function, as long as it is differentiable. The system of equations 2.2 and 2.3 constitute the entire dynamics of all units in the network $\mathcal{N}$.

Training a neural network involves finding the set of weights for which the outputs are as close as possible to the desired targets given the input data, so as to minimize the total error over all examples. This process could be performed by investigating all possible combinations of weights and picking the combination that minimizes the total error. One can imagine that trying all possible combinations of the weights is computationally intractable. Therefore, most training algorithms do not search for the minimum error by trying all possible combinations of weights, but try to search for the minimum error in a more intelligent way. For example, the backpropagation algorithm tries to find the minimum error by updating the weights into the direction of the negative derivative of the error with respect to the weights. This process can be visualized by imagining the weights and the error as a multidimensional space, in which each weight represents one dimension, and the value of the error one additional dimension, as depicted in figure 2.3. The algorithm searches for a minimum of the error in the weight space using an iterative gradient descent method. In each step of this method, the direction of the negative derivative of the error with respect to the weights is determined, and the update of each weight is the proportion of that weight to this derivative. The combination of weights that minimizes the error function is considered to be a solution of the learning problem. Because this method uses the derivative of the error function and the error function is computed in each unit, the components of the network must guarantee the continuity and differentiability of the error function. This means that the summation and the transfer function of each unit should have those properties. A function that is often used as transfer function is the sigmoid, because it is a non-linear function, and its derivative is quite easy to compute:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2.4}$$

and its derivative:

$$f'(z) = f(z)(1 - f(z)) \tag{2.5}$$

Backpropagation with gradient descent tries to find a minimum of the error function by determining the direction of the partial derivative of each weight with respect to the error function:

$$\nabla E(\vec{w}) = \left(\frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n} \right) \tag{2.6}$$

This process is visualized in figure 2.3. Because the number of dimensions that can be depicted is limited, this figure shows only two weights w_1 and w_2, and the error e, which is computed by subtracting the target value t, from the actual output y. The lines in the figure determine the landscape of the error surface with respect to the weights. Now imagine that the two weights have certain values whose location in the error surface is given by point b. The direction of the total weight change that decreases the error is given by the direction of the black arrow, which points in the direction of a, the global minimum. The direction in which the two weights should be changed is given by the gray arrows, which is the direction of the negative partial derivative of each of the weights with respect to the total derivative.

The example in figure 2.3 is of course very simplistic in that more realistic networks have more weights and therefore more dimensions. The error surface in such a high-dimensional space may not be as smooth and regular as in the example, and may contain more local minima. Moreover, there is another problem with gradient descent. As can be seen in figure 2.3, the negative derivative of the error surface does not always point in the direction of the global minimum. If, for example, the two weights have certain values whose location in the error surface is given by point c, the direction of the negative derivative points away from the global minimum. This means that backpropagation with gradient descent is not always guaranteed to find the global minimum, and depends on the initial values of the weights. Yet another drawback of gradient descent, is that it learns very slow. Each update changes the weights with a certain amount, called the *learning rate*. The learning rate can be a number between 0 and 1, but it has to be chosen carefully. If it is too big, the process will overshoot the minimum, and if it is too small, the learning process is very slow, and can be stuck in local minima. Even with an appropriate learning rate it usually takes a large number of training cycles to find a minimum. A possible solution to this problem is to use the second derivative of the error function or an approximate as implemented in the Levenberg-Marquardt algorithm (Hagan and Menhaj, 1994), or Quasi-Newton methods (Dennis and Schnabel, 1983). In this project however, gradient descent with the first order derivative will be used, so we limit ourselves to describing the computation of the first derivative.

The process of finding the direction of the weight changes for each separate weight can be given by the mathematical description of the backpropagation algorithm. The set of output units T, contains the units with indices $i \in H$, for which there exists a specified target value d_i, that the output y_i, of unit i should match. The error e_i, in each unit is computed by the equation:

$$e_i = \begin{cases} d_i - y_i & \text{if } i \in T \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

Note that although the error is computed in each unit, only the output units $i \in T$, can have a value other than 0. The total error over all output units is defined as:

$$E = \frac{1}{2} \sum_{i \in T} [e_i]^2 \tag{2.8}$$

The objective is to minimize the total error over the output units. This can be established by propagating the error at the output units back through the network, computing the local error for each unit in terms of the proportion of the total error E, to the input of the unit net_i:

$$\delta_i = \frac{\partial E}{\partial net_i}, \tag{2.9}$$

for all $i \in H$, which is:

$$\delta_i = f_i^{'}(net_i)[e_i + \sum_{j \in H} w_{ji}\delta_j] \tag{2.10}$$

Equation 2.10 holds for indices $i \in T$, of units for which a target value is defined, as well as indices $i \notin T$, of units that don't have a target value. In case there exists a connection w_{ji}, with $i \in T$, and $j \in H$, the local error of the unit is added to the back-propagated δ_j from the connected unit with index j. The gradient descent training algorithm computes the updates for the weights according to:

$$\triangle w_{ji} = -\epsilon \frac{\partial E}{\partial w_{ji}} = \epsilon \delta_j y_i, \tag{2.11}$$

with positive learning rate parameter ϵ. In this way each weight is updated by propagating the δ's from the top layer back towards the input layer, thus updating each weight with its proportion to the derivative of the error.

2.4 Recurrent neural networks

An important aspect of feedforward neural networks as discussed in the previous section, is that units in a certain layer receive only inputs from units in a lower layer. This simplification of the actual structure of biological neural networks results in some very nice mathematical properties of feedforward networks, such as the possibility to use a gradient descent algorithm based on the top-down propagation of the error. The drawback is that it requires the input patterns to be presented in separate snapshots. Non-recurrent neural networks cannot employ the dependencies *between* different snapshots, and thus cannot learn to find relations in time series data. If the simplification of having no recurrent connections is dropped, and units can have input connections from units in higher layers or even from themselves, an additional temporal dimension is added to the network. An example of an artificial neural network with recurrent connections is given in figure 2.4. Each time an input pattern is presented to the network, the units compute their activation just as they would have done in a feedforward network. However, the inputs to the unit now contains a term which reflects the state of the network before the pattern was seen, through the activation of units in the previous step. When subsequent patterns are presented, the states of the units in the network will be a function of everything the network has seen so far, giving the network a sense of history.

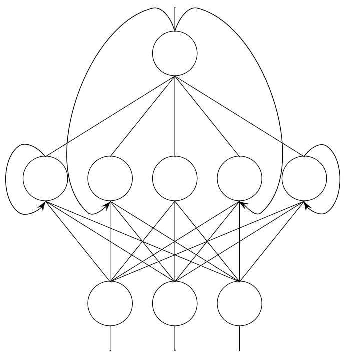


Figure 2.4: Example of a network with recurrent and self-recurrent connections

In most recurrent neural network architectures, the computation of the states of the units takes place in discrete time steps. At each time step, the activation propagates forward through one layer of connections only. This ensures that the activation of each unit that provides input to another unit is known at the next time step. Furthermore, the error can still be propagated back through the network, while taking into account the extra temporal dimension. Once some level of activity is present in the network, it will continue to flow through units, even if no further input is presented. Instead of providing the input patterns as snapshots, it is now possible to present a network with a time series of inputs, and also require the network to produce output based on this series. This results in a whole new set of problems that can be addressed by recurrent neural networks, but also generates certain additional difficulties while training these networks.

In a recurrent neural network, information can be stored in two distinct ways, a short-term memory and a long-term memory, analog to models used in computer science and psychology (Atkinson and Shiffrin, 1968). The activation of a unit is a function of the history of the states of the network, thus forming a sort of short-term memory which is lost when the activities in the network are reset. The weights of the network also implement a sort of memory as they are trained on past experience, but the timescale of the weight changes is much longer than that of the activations. The weights of a recurrent neural network can thus be seen as a form of long-term memory.

The reason why the short-term memory that is based on previous activities in the network is really short, is that it is very difficult to retain a certain stable level of activity. Once a certain level of activation is reached in the network, connections with a weight larger than 1 will cause the activation to increase exponentially, and connections with a weight less than 1 will cause the activations to decrease exponentially. Non-linear activation functions of the units will prevent the activities from growing without bound, but the problem is that it is very difficult to train the network to use of this sort of memory (Hochreiter and Schmidhuber, 1997). As the error is propagated back through the network, the contribution to the error of a particular weight situated lower in the network becomes more indirect. The error is propagated back over time, and each propagation dilutes the error until far back in time, it is too small to have any influence. This makes it very difficult to assess the importance of network states at times that lie far back in the past. Using the short-term memory that is present in the activations of the weights back in time, is therefore very difficult. In general, recurrent neural networks that are trained by gradient descent methods such as Realtime Recurrent Learning (RTRL) and Backpropagation Through Time (BPTT) (Williams and Zipser, 1989; Rumelhart et al., 1986) cannot reliably use states of the network that lie more than about 10 time steps in the past. In the literature, this problem is often referred to as the problem of the *vanishing gradient* (Hochreiter, 1998; Bengio et al., 1994; Pearlmutter, 1995; R.J. and Zipser, 1995; Lin et al., 1996).

Chapter 3

The liquid state machine

3.1 Problems with recurrent neural networks

As discussed in the previous chapter, recurrent neural networks are more powerful than neural networks without recurrent connections, because the recurrent connections allow a network not only to process input patterns over time, but also employ these connections to maintain some sort of short-term memory. Non-recurrent neural networks can only handle snapshots of situations as input. For each input pattern a target has to be specified, and the computed error for that snapshot is used to adjust the network as to minimize the error on future inputs. This process is unlike what is going on in real biological neural networks, such as the human brain. The brain is constantly supplied with a large amount of rapidly changing information. Even in the case the brain is supplied with a still picture, for example a photograph of a human face, the visual information is not presented to the brain in a single snapshot, but as the eyes of the observer look at different parts of the picture, more information of the observed face becomes available over time. To recognize a whole face from the observed parts, it is crucial that some kind of memory is present to relate current inputs to former ones, in the parts of the brain that are involved in facial recognition. In the case of artificial neural networks it is of course possible to increase the number of input units, such that several snapshots can be provided to the network at once. It is however not always a priori known over how many time steps the temporal dependencies span, which makes it difficult to determine how many patterns should be presented to the network at once. Besides, in case the dependencies span a very long time, the network becomes very large, which results in low learning speeds and increased chance of overfitting. Whereas neural networks without recurrent connections cannot use any information that is present in the dependencies between inputs that are presented at different time points, recurrent connections allow a network to keep a short-term memory of previously seen inputs. To model biological neural networks and to employ the enormous computational power of such networks, neural networks that can only process snapshots of situations will not suffice.

However, a major problem with recurrent neural networks is the training of such networks with supervised learning. The types of neural networks discussed in the previous chapter are already very simplified models of biological neural networks, but they still have to cope with the problem of vanishing gradient. Trying to model a real biological neuron with an implementation that describes exactly what happens when which chemical influences the membrane potential, might result in a very detailed and complex simulation, but is usually accompanied by a large computational cost. Additionally, the more complexity is added to the model to closer match real biological structures, the more difficult it gets to train such a model with supervised learning methods. As in most experiments the electrochemical processes in individual neurons are not subject of the study, but rather the total complex behavior that a neural network can model, it is not always necessary to model the complexity of a real neuron to its full extent.

One artificial neural network architecture that is more like biological neural networks, is the recurrent neural network with spiking units. This architecture is more closely related to its biological counterpart than feedforward networks, because the units in this network do not communicate between each other with continuous values between 0 and 1, but by sending discrete action potentials. Just as in a biological neurons that send out short pulses of electrical energy as a signal, in this model communication between units is based on individual action potentials or *spikes*. When a spike arrives at a unit, it increases or decreases the membrane potential, which then slowly returns to its resting potential. When a number of spikes that arrive shortly after each other have increased the membrane potential above a certain positive threshold value, the unit itself sends a spike to other units. This mechanism quite differs from the mechanisms by which feedforward networks and recurrent versions of the feedforward networks operate, but neural networks with spiking units are capable of showing extremely rich and dynamical behavior with a relatively small sensitivity to noisy input.

In contrast to the neural networks discussed in the previous chapter where the output of a unit is represented as a real number, the activity of the units in a spiking neural network often makes no sense to humans. An important question is how the information that is present in the individual spikes and their precise timings is actually encoded, or how it should be represented. This problem has been the subject of an ongoing debate in the field of spiking neural networks. Biological neurons send in some sense binary outputs: they send an action potential, or they don't. In artificial neural networks this way of communicating information between units is however not common practice. For example, the units in the feedforward networks as discussed in the previous chapter, do not employ individual spikes, but use an interpretation of this neural code in the form of average firing rates. The input of a unit from another unit does not consist of individual action potential equivalents, but is a number between 0 and 1, representing the number of times the unit sends a spike during a certain interval. This interpretation with averaging the number of spikes, not only enables computer simulations to be calculated in iterations, but also makes it possible to adjust the synaptic strengths or weights of connections between units to be adjusted according to powerful gradient descent algorithms.

In biological neural networks there is no evidence that a neuron uses some kind of averaging mechanism to interpret incoming signals into meaningful output signals. Research has shown that even complex tasks such as human facial recognition can be performed in very short time, which makes it unlikely that neurons use an average rate coding (Thorpe et al., 2001). Therefore neural networks with spiking units need not to interpret signals between units as average firing rates, but can employ the information that is present in the timing between individual spikes. Much more information can be sent in a stream of individual pulses than can be transmitted using just averages of pulses, thus creating a whole range of possibilities for new interpretations of neural code. Several neural codes that could be used for communication between units have been proposed by Thorpe et al. (2001), such as binary code which interprets a spike as a binary 1 and no spike as a binary 0, timing code, which takes into account the precise timing between individual spikes, rank order code, in which information is stored in the temporal and spatial order of incoming spikes, and synchrony code, in which the amount of units that fire at the same time is taken into account. The latter two are more realistic because the amount of information that can be sent in such ways is maximized, but depending on the ease of implementation of these possible neural codes and the purpose of the simulation, several possibilities and all sorts of combinations can be used.

In a neural network with spiking units the chance that a certain unit fires at a certain time t, is very small, because the time of a spike is very small compared with the other time constants. If the state of such a network at a certain time is needed for further processing, not much information can be acquired from just taking a snapshot of the activities of the unit at t. Therefore the average firing rate over a certain interval of a group of units is often used instead of individual spikes of units. This interpretation of the activity in the network, called population rate code, is often used, not to communicate signals in the network, but to extract information from the network for further processing. This kind of neural code is unlikely to be biologically plausible because the amount of neurons that would be required for difficult tasks is not present in the human brain (Kandel et al., 1991). It is however very easy to implement, and can be used to extract information from spiking neural networks without imposing a neural code that should be used *inside* the network.

While Hebbian learning might be useful for some tasks that can be learned by unsupervised methods in networks with spiking units, supervised learning of time series data with such units is even more difficult, as it involves controlling the responses of highly non-linear and time-sensitive units. For most neural codes, adjusting the weights of the connections such that a certain error function is minimized, is already very difficult, but as there is no clue which code is actually used in the brain it is not clear which one to use anyway. Processing the rapidly changing information that enters the human brain is performed by intricate organized connections between different neurons, that form an immensely complicated network with multiple recurrent loops. Modeling these structures such that learning processes produce sensible results, turned out to be intractable in the past, and still is nowadays (Savage, 1998). It might however not be the case that there is an actual neural code that is used in biological neural networks that needs to be understood to control the behavior and learning in neural networks, but that the power of spiking neural networks lies in the dynamics. As long as such a network produces a certain behavior that is complex, it is not necessary to steer and control the network dynamics by adjusting the weights of the connections by more or less ingenious learning algorithms.

3.2 The liquid state machine

Instead of training a recurrent neural network to learn to exploit memory over previous states, the liquid state machine (Maass et al., 2002) is based on the idea that the dynamics of spiking neural networks can be employed as a temporal filter. At any moment this liquid filter contains information on its present and past inputs. The state of this liquid can be read out and presented to a separate readout mechanism at certain times. This readout mechanism can be trained to learn to extract useful information from the state of the liquid, while the liquid itself doesn't need to be trained. As an illustration for this approach, imagine a series of perturbations in a liquid, such as water, caused by a sequence of external disturbances, for example by raindrops falling into the water. The water will respond by producing waves and ripples which travel through the water and over its surface, creating specific resonating patterns. The successive states of the water, or the *state trajectories*, resulting from a disturbance will be different for each individual raindrop. At any moment the state of the liquid contains information on present as well as previous disturbances that were applied to it. A readout mechanism, such as an artificial neural network, can be trained to match its output to a certain target given the state of the liquid at certain times as input. It is not necessary anymore that the readout mechanism implements memory since the information on previous inputs are reflected in the state of the liquid at any moment.

Although the idea for the liquid state machine originates from the dynamics of spiking neural networks, it can be applied to any liquid that shows some kind of dynamical behavior. In order for a liquid to serve as a source of salient information for the readout mechanism, the perturbations should vary for different inputs, but the behavior of the liquid should not become chaotic, since little information can be extracted from total chaos. Whether this is the case or not for a certain liquid, depends on the complexity and dynamics of the liquid. In general, more complex liquids with more degrees of freedom of the processes that govern its behavior, produce more diverging internal states, but also have a greater risk of showing chaotic behavior. The property of a liquid that it produces increasing divergence of internal states over time caused by different input patterns, is called the *separation property.*

As in most learning tasks it is not required to retain information on all previous inputs infinitely long, the dynamics of the liquid are required to slowly ‘forget’ about inputs that lie long in the past. In the illustration with raindrops falling into water, this behavior corresponds to slowly fading waves and ripples due to friction. The information that can be extracted from the form and location of the ripples in the water surface slowly becomes less over time, and when a new drop falls into the water the contribution to the ripples and waves of the latter is more than the first. This property of liquids is called the property of *fading memory* (Boyd and Chua, 1985).

Any system that satisfies the properties described above can be used as liquid in a liquid state machine. Moreover, the liquid state machine does not require a task-specific constructed liquid as long as it has the properties of a dynamical system with sufficient degrees of freedom. Different readout mechanisms can even be trained to perform different tasks on the same state trajectories of the liquid, enabling parallel computing. The way in which perturbations in the liquid are formed and propagated, varies for different types of liquids and strongly determines how useful it is. The computational capabilities of liquids are determined by their time to relaxation, local or non-local interactions, and the extend of heterogeneity of the elements or particles of a liquid.

It is not required for the liquid to have stable internal states representing features in the input that evolve over time, moreover it might not even be the case that a certain state occurs more than once. If the liquid has the properties of a dynamical system with sufficient high dimensionality, the readout mechanism can transform the transient states of the liquid to stable target outputs. The capability of the readout mechanism to distinguish and transform the different internal states of the liquid to match the given target, or the *approximation property*, depends on the ability to adjust to the required task of the readout mechanism. The readout mechanism should be able to learn to define its own notion of equivalence of dynamical states within the liquid, despite the fact that the liquid may never re-visit the same state.

In contrast to the liquid, the readout mechanism has to be constructed in a task-specific manner in terms of the desired output. Memory on previous inputs is accomplished by the liquid, so in general the readout mechanism can be memoryless, which makes it much easier to construct a task-specific readout mechanism. Unlike finite state machines where the internal states are already constructed in a task-specific manner, the liquid state machine does not need to store information about the past in stable states, such as memory buffers or delays. As long as the dynamics of the liquid ensure significantly different internal states of the liquid over time for two significantly different inputs, it is not important to store information about previous inputs in stable states. That means that it is not required to carefully define the states and transitions as is done in a task-specific finite state machine; the liquid state machine depends only on identifiable internal states caused by an input pattern.

Figure 3.1(a) gives a schematic overview of the liquid state machine. Let $\mathcal{M}$ be a liquid state machine with liquid $L^{\mathcal{M}}$ and readout mechanism $f^{\mathcal{M}}$. The input to the liquid is a series of disturbances $u(\cdot)$, the outputs of the readout mechanism is some series $y(\cdot)$, and the targets are denoted by the set $d(\cdot)$. $\mathcal{M}$ produces at every time t, an internal liquid state $x^{\mathcal{M}}(t)$, which represents the current response to previous perturbations $u(t')$, where $t' < t$. The task of this liquid state machine is to map inputs $u(\cdot)$, to outputs $y(t)$, by adjusting the readout mechanism so that $y(t)$ matches $d(t)$ more closely for a given internal state of the liquid $x^{\mathcal{M}}(t)$. The liquid can be seen as some temporal filter $L^{\mathcal{M}}$, that maps inputs functions $u(\cdot)$, onto functions $x^{\mathcal{M}}(t)$, and the readout mechanism as a memoryless readout map $f^{\mathcal{M}}$, that transforms the liquid state

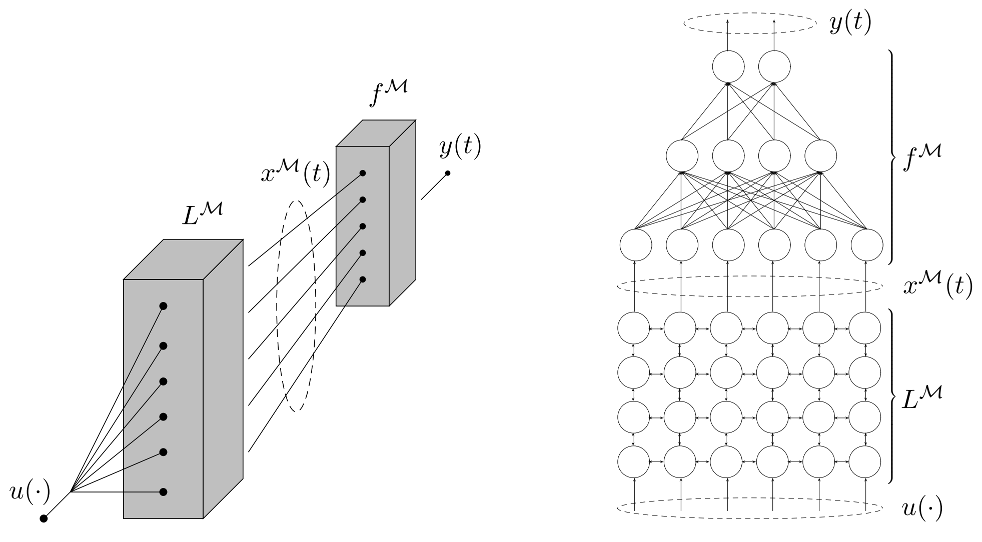


Figure 3.1: Schematic architecture of the liquid state machine (a), and its neuronal implementation (b)

$x^{\mathcal{M}}(t)$, into an output $y(t)$, at every time t. This general description specifies the components that are necessary and sufficient to build a liquid state machine, as long as these components satisfy the properties described before.

Injecting time-varying input patterns into a liquid with sufficient complex dynamics, will in most cases not reveal any information on the nature of the input patterns for the human observer. It is however crucial to the efficacy of the liquid state machine that the readout mechanism can discriminate between different state trajectories caused by the input patterns. One reason this is the case for a whole range of liquids, is that the preprocessing that is performed by the liquid can be viewed as the projection of the input patterns in a high-dimensional space. The decision surface which determines whether the liquid is in a certain state at some specific time, can be approximated by a hyperplane in the state space of the dynamical system that governs the behavior of the liquid. Statistical learning theory tells us that learning becomes easier when the input patterns are first projected into sufficient high-dimensional space (Vapnik, 1998). Due to the high-dimensionality of the space of the liquid, the projections of the original input patterns become linearly separable. In general, this means that the more degrees of freedom the liquid has, usually the easier it becomes for the readout mechanism to learn to predict the right targets. Whereas methods used in statistical learning theory do not cast the input patterns into a high-dimensional space explicitly, this process can in liquid state machine be attributed to the essential mechanisms that govern the behavior of the liquid.

Maass et al. (2002) prove that liquid states machines that satisfy the separation and the approximation property have universal power for computations with fading memory on functions of time. Häusler et al. (2003) argue that this universal computational ability is due to the non-linear projection of the input patterns in high-dimensional space, which is performed in the liquid. Neither this idea, nor Maass' original theory of the liquid state machine makes any specific requirements with regard to the structure of the components of the liquid. As long as it satisfies the aforementioned properties, any system can successfully be employed, even real water (Fernando and Sojakka, 2003).

3.3 Neural circuitry as liquid

Recurrent neural networks with spiking units do not only have a large variety of mechanisms and constants determining their interactions, but can also have local as well as non-local interactions. This means that they can effectively be used as high-dimensional liquid equivalents. Maass et al. (2002) used a simplified model of the cerebral cortex, in which the individual neurons are organized in columns. These columns in the cortex are all very similar and can perform many complex tasks in parallel. Not only are these heterogeneous column-like structures found in the human cortex; they are also present in the cortex of other species. Implementation of these columns in the form of a spiking neural network, have already shown to perform very well, and even in noisy environments their dynamics remain reliable (Vreeken, 2004). Therefore the liquid in this project will be a recurrent neural network with spiking units, organized in a column-like structure.

The spiking neural network tries to capture the behavior of real biological neural networks, but does not necessary implement all features and interactions of its biological counterpart. One model that is commonly used for networks of spiking units, is the integrate-and-fire model (Feng and Brown, 2000). This is a general model that implements certain properties of biological neurons, such as spikes and membrane potentials, and describes in basic terms that the membrane potential of a unit slowly leaks away, and that a unit fires if the membrane potential reaches a certain value. This model is rather simple to understand and implement, and can be described by giving the differential equations that govern the behavior of the membrane potentials, spikes and some other variables over time.

In the model used in this project the states and variables of the units are computed in discrete time, using a universal clock that updates the values of a unit each time step. Therefore the equations that describe the behavior of such units will not be given in differential equations, but are described in terms of the updates of the membrane potentials and activities of the units, that are computed iteratively each time step t. The units used in this simulation are a simplified model of their more realistic biological counterparts. Spikes or action potentials are modeled by short pulses that are communicated between units. The input signals to a unit i are used to increase or decrease the membrane potential equivalent μ_i. Once μ_i reaches a certain threshold value η, the unit sends a signal over its output to the inputs of other units.

Action potentials in biological neurons are all very similar. Therefore all spikes are modeled to have the same value, so the output y_i, of unit i is 1 if a unit fires, and 0 if it does not fire. Most of the time a unit does not fire, meaning that its output is 0 and does not contribute to the input of the units it is connected with. The incoming pulses are multiplied by the synaptic strength equivalent, or weight w_{ij}, of the connection between the output of unit j and the input of unit i, which can be a positive or negative value to model excitatory or inhibitory connections respectively. Because the effect of an incoming signal lasts longer than the duration of the pulse itself, and the membrane potential at the postsynaptic terminal of a biological neuron slowly returns to the resting potential, the accumulated input of a unit at each time step is added to a proportion of the membrane potential of the previous step. The parameter that controls the speed at which the membrane potential returns to its resting potential is:

$$\tau = \left(\frac{1}{2}\right)^{\frac{1}{r^h}} \tag{3.1}$$

where r^h is the number of time steps in which half of the membrane potential is left if no additional inputs are presented during this time. The membrane potential μ_i, of unit i is computed each time step according to:

$$\mu_i(t) = \tau \mu_i(t-1) + \sum_j w_{ij} y_j(t), \tag{3.2}$$

Subsequently the output, or whether the unit fires or not, is computed by comparing it with the threshold value η:

$$y_i(t) = \begin{cases} 1 & \text{if } \mu_i(t) \geq \eta \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

After sending an action potential, a biological neuron cannot fire again for a certain period, followed by a period in which it is more difficult to fire. This behavior is modeled in our artificial neurons by implementing an absolute and a relative refractory period. After sending an action potential, it is impossible for a unit to fire again for a number of r_i^a time steps, where r_i^a is the absolute refractory period of unit i. Combining this with equation 3.3, yields:

$$y_i(t) = \begin{cases} 1 & \text{if } \mu_i(t) \geq \eta \text{ and } (t - t_i^f) > r_i^a \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

where t_i^f is the last time unit i generated an action potential. When the absolute refractory period has ended, a relative refractory period is simulated by resetting the membrane potential μ_i, to some negative value r_i^r, where $-r_i^r$ models the relative refractory period of unit i, thus making it harder for the membrane potential to reach the positive threshold value η. To implement this relative refractory period, equation 3.2 is augmented with an additional term:

$$\mu_i(t) = \begin{cases} \tau \mu_i(t-1) + \sum_j w_{ij} y_j(t) & \text{if } (t - t_i^f) > r_i^a \\ r_i^r & \text{otherwise} \end{cases} \tag{3.5}$$

In case there is no additional input presented to the unit, the membrane potential will slowly return to the resting potential, just as it would for a positive membrane potential in case no additional inputs are presented, according to equation 3.2. Equations 3.4 and 3.5 sum up the whole of the dynamics of spiking units of which the spiking neural network consists.

3.4 Readout mechanisms

Input patterns that are presented to the liquid cause certain state trajectories in the liquid, which reflect at any moment information on present as well as past inputs. As discussed in section 3.2, there is a single factor that determines whether the readout mechanism can extract information from the liquid states; the approximation property. This property formalized by Maass et al. (2002), states that a class of functions that satisfy the approximation property, always contains a function that produces a different output than any other continuous function, given all possible inputs. This might seem a little abstract, but in other words this means that this class of functions is guaranteed to contain a function that produces a unique value for each possible input. The task of the readout mechanism is to find a function that can generalize over different inputs, and also produces the right targets. In particular, the readout mechanism should learn to define its own classes of equivalence of dynamical states within the liquid, so that it can perform this task on inputs it has never seen before. This enables such a mechanism to perform its task despite the fact that the liquid may never re-visit the same state.

Häusler et al. (2003) argue that as the liquid state machine operates by the principle of projecting the input patterns in a high-dimensional space, very simple and robust readout mechanisms should be able to learn very well. Reasonable results have been obtained by Häusler on toy problems, using very simple readout mechanisms, such as single readout units trained with the backpropagation algorithm, or linear perceptrons. The high dimensionality of the liquid seems to be the key factor, as this enables a sufficiently large number of degrees of freedom for the readout mechanism to extract individualized states of the liquid. As the training algorithms that are used in these methods hardly suffer from being stuck in local minima, this seems to be a very efficient readout mechanism. However, some assumptions are being made which are problematic for the biological interpretation of this framework.

First, the liquid, consisting of the spiking neural network described in the previous section, should be rather large. This decreases the probability that the learning algorithm of the readout mechanism will be stuck in local minima by ensuring the necessary degrees of freedom in the liquid from which the readout mechanism can extract individualized states. Although the human brain consists of a very large number of neurons, it is improbable that the enormous amount of neurons is available that would be required to perform such a range of different tasks by just projecting the sensory input into a high-dimensional space, and using simple linear readout mechanisms for learning (Kandel et al., 1991).

Second, the readout mechanism and the spiking neural network use different 'languages'. Therefore some sort of translation has to be implemented. Spiking neural networks as discussed in section 3.1 communicate using individual spike times, whereas the readout mechanisms usually applied in liquid state machines are based on rate coding. Transforming the states of the spiking neural network seems quite problematic at first sight. Because the units are not firing most of the time, the activity in the network is sparse, and almost no information can be supplied to the readout mechanism. Another possibility is not to use the activity of the units, but the membrane potential. However, because of the highly non-linear behavior of these potentials, they do not provide more information than the separate spikes (Buonomano, 2000). One solution that is often used for extracting information from spiking neural networks, is to count the number of times a unit fires during a certain interval. To make this method more reliable, this process is performed for a number of units in the liquid and the results are averaged. In this way all information that is available through the temporal aspect of the spikes is neglected. This seems a major drawback, but as the internal dynamics of the liquid is not affected by this process, this method often still produces reasonable performance.

The third assumption is not an assumption of liquid state machines specifically, but of supervised classification in general. In classification problems, input patterns that are provided to the network belong to a certain class. The task of the network is to indicate the right class for a given input pattern. In the liquid state machine, the input patterns are injected in the liquid. The disturbances from the resting state advance through the liquid and eventually affect the whole liquid. The liquid state is used to extract information on the present and previous inputs, and is then transformed by the readout mechanism to a certain class that corresponds to the input pattern. In most readout mechanisms as described by authors who previously used the liquid state machine (Maass et al., 2002; Vreeken, 2004; Häusler et al., 2003), the total distributed state of the liquid is transformed into the local representation of the activity of one unit in the readout mechanism. Take for example a speech classification task of a person saying 'cat' or 'dog'. The task of the readout mechanism is to activate unit 1 if the input to the liquid was 'cat', and unit 2 if the input was 'dog'. In this example the concepts 'cat' and 'dog' are explicitly represented in the specific units that are pre-assigned to do so. From the perspective of biological neural networks it is highly unlikely that this modeling choice reflects realistic biological processes (Churchland and Sejnowski, 1992).

It is beyond the scope of this project to elaborate on the symbolism versus connectionism debate or distributed versus localized representations. The purpose here is to investigate the use of an alternative readout mechanism, that is based on different principles not necessarily involving the assumptions and drawbacks of other readout mechanisms as discussed before. This alternative readout mechanism is the Helmholtz machine. The Helmholtz machine is a connectionist system that is based on building probabilistic models of the outside world, and uses this model to recognize novel input patterns. Model building is accomplished by finding economical descriptions of the features that are present in the input patterns. As the Helmholtz machine can be trained unsupervised, and uses distributed representations, this would be a more biological plausible candidate for a readout mechanism of the liquid state machine than the models used by others.

Chapter 4

The Helmholtz machine

4.1 Introduction to pattern theory

Since the beginning of neural networks research, the focus has largely been aimed at forward processing of incoming information. In this view incoming information is transformed to concepts which represent certain features, and these concepts can be used to form even higher-level concepts. This idea might not seem very strange, as to humans it seems that sensory information is effortlessly and instantaneously transformed into concepts we perceive, such as a spoken word or a face. Most classification methods used nowadays, such as feedforward neural networks, parametric statistical modeling or tree-based classifiers, are based on the idea of transforming incoming information into concepts that can be used for further processing. Mumford (1994) radically breaks with this idea and describes a new view on classification based on evidence from the field cognitive psychology, information theory and statistical physics. The most important difference with other views is that classification does not only involve the transformation of signals to concepts, but also the *reconstruction* of what the signal is like. Applying this idea to the interpretation of the processes that are going on in the human brain, implies that the noisy and ambiguous signals that are picked up by the sensory system never make it to the consciousness, but are replaced by more ideal representations that are created with extensive use of memory and expectations.

The key idea in Mumford's theory, called *pattern theory*, is that sensory information gives a clue to what is really going on in the outside world, and that processing these clues involves reconstructing as good as possible the real state of the world. Sensory information is acquired through available patterns in the outside world. These patterns are caused by underlying processes. The task of recognizing patterns is to reconstruct the processes that caused the perceived patterns to occur. Take for example a pattern of auditory signals produced by speaking a certain word. When this pattern reaches a human ear, the first step in recognizing the word that was spoken, is to reconstruct the events in the throat and mouth of the speaker, and to connect these events with the specific linguistic properties that can be used for further processing. Pattern theory however, is not limited to reconstructing processes at the level of sensory input; it can also be applied to higher-level concepts. An example of this is a medical expert system in which the input to the system is a list of symptoms and test results. This information was of course generated by the processes in the patient's body, and the task of the expert system is to reconstruct the processes given these clues. Pattern theory considers the outside world as the source of unknown processes, objects and events that produce encoded patterns. A classification model can classify this information into sensible concepts by reconstructing the processes in the outside world that caused these patterns to occur.

To perform this task, pattern theory gives some basic properties that should be present in models that implement this theory. Although these properties are not restricted to neural networks, pattern theory as described by Mumford is mostly about neural networks, so the description of these properties will be restricted to the application of neural networks. To enable the synthesis or reconstruction of the input patterns, a simple feedforward architecture will not suffice, because the reconstruction of input patterns can only be achieved by feedback, top-down connections. The process of sending information over these feedback connections, as well as the feedforward connections, cannot be completely arbitrary, because the architecture needs to encode the transformed input patterns in a certain way to be able to reconstruct them. Reconstructing the input patterns must be learned by the network through experience. As more information of the outside world is presented to the network, it learns to build a model of the outside world that can be used to synthesize input patterns. Just as in most orthodox feedforward neural networks, models that operate on the principle of pattern theory can also have a hierarchical layered structure. Evidence for such layered structure that has feedforward and feedback connections, is found in the human cortex (Felleman and Essen, 1991). Although there is no strong evidence that the human brain operates on the principles of pattern theory, the structure of the cortex in particular matches the properties described before. Some evidence however comes from the field of cognitive psychology, where a large number of experiments has shown that what we perceive is not what is actually there, but rather a reconstruction from our imagination depending on our memories and the situation.

Learning in pattern theory is largely rooted in Bayesian statistics. The processes that describe what is going on in the outer world can be described by certain variables Θ. These processes are not directly known to the model, and are therefore called the hidden variables. The patterns that are created by the processes are the data that can be observed, and will be denoted by the set of variables U. From previous observations the model knows for every possible subset $\theta \in \Theta$, the prior probability $p(\theta)$, that it occurs in the outer world. Furthermore the model learns the probability of each relation between the possible input patterns and the real-world processes that caused them: $p(u|\theta)$. Presented with a certain input pattern u, the task of recognizing this pattern consists of selecting the most likely posteriori values of θ that caused u to occur. According to Bayes (1764), this task can be performed by choosing the θ which makes the conditional probability $p(\theta|u)$, the highest. The model learns by applying Bayes theorem on both the prior probability distribution on the outer world processes Θ, and the conditional probability on the observations U, given Θ. The method that is used in pattern theory, is an application of the minimum description length principle. The minimum description length principle can be used to re-code the input patterns in a form that requires less information by finding regularities in the data. These regularities are often caused by the same processes, so it is by using the minimum description length principle that the underlying processes that generated the observed data can be found.

Mumford gives some ideas on how this could have been implemented in the cortex, and also speculates about the evolutionary origin of this architecture in the mammalian cortex. His hypothesis is that the probabilities of the set of variables of hidden world states and processes are stored in a mechanism that generates patterns in such a way that the patterns that are more likely to occur in the outer world, are also more likely to be generated. As more information comes available and is processed, the better the synthesized patterns can be matched with the actual input patterns. Patterns that are more likely to be generated can for instance be inhibited by the input signal, resulting in the synthesis of a less likely pattern from the generative mechanism. The task of the cortical structure is thus to find the simplest representation that will generate a synthesized pattern through the top-down connections and is as close as possible to the true input pattern. Because the patterns in the outside world are often fuzzy and distorted by noise, it is usually not possible to regenerate the exact input pattern. Therefore the model has to learn more idealized versions of the input patterns that are general enough to deal with noise. For example, the concept of ‘cat’, or ‘dog’ is an idealized and general concept which a model has built from several noisy instances of cats and dogs, without ever having seen an ideal cat or dog. Elaborating on the idea that humans build idealized versions of concepts that are close to the true underlying processes that generated the observations, it might be the case that the very math that is used to describe the fundamental processes in nature is a result of this mechanism.

Pattern theory is quite different from more orthodox methods in the field of artificial intelligence. Whereas the latter methods deal with strict rules, concepts, and formal analysis of processes, the ideas of Mumford fit better with non-logical data structures such as probabilities, fuzzy sets and noisy data. The basic idea of pattern theory that in order to recognize patterns, it is also necessary to be able to synthesize them, clearly distinguishes this theory from other methods in the field of artificial intelligence. It implies that top-down or feedback processes are just as important as bottom-up feedforward ones. The problem of just transforming input patterns into concepts with certain features, without being able to recreate the input pattern given certain features, is that incoming patterns are not explicitly modeled. In pattern theory it is more important that a part of the input pattern matches the class, than it is for all parts of it to match slightly. This makes classification more robust, and less sensitive to noise. In more orthodox methods where this top-down synthesis is not employed, this is not possible.

4.2 Processes, patterns and models

Mumford's theory states that classification, as is for example performed by the cortex, is not only about representing the input patterns efficiently, but also about building probabilistic models for those patterns by finding the structure that is inherent in the set of patterns. One possibility to do this, is to build a parameterized stochastic model that can reproduce each pattern from an independent draw, according to the same probability distribution as the observed set.

An example of a model that generates such patterns is depicted in figure 4.1 (example adapted from Dayan et al. 1995). Each pattern consists of a 4 × 4 pixel bitmap. The patterns are generated in two steps. In the first step each pixel in the top row is turned on with a probability of 0.2. Subsequently the top row is copied to the bottom row, and shifted one pixel to the left or right with wraparound. The two middle rows are then copied from the adjacent outer rows, introducing redundancy into the images. Detecting the direction of the shift resembles the task of extracting depth from stereo images of vertical line segments, where the top two rows are the left retina, and the bottom rows the right.

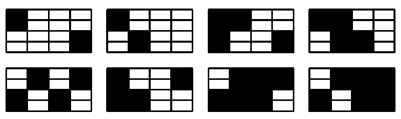


Figure 4.1: Example of patterns created by a generative model

In this example, the task of learning is to induce a model that infers the probability distribution over the underlying causes of the observed patterns generated by the two-stage hierarchical process. Even in this very simple process, any pattern can have multiple simultaneous causes, whereas more realistic problems can even have an exponential number of possible causes underlying each pattern. Training a model from a set of patterns generated by a more realistic process, involves inference of an exponential number of possible causes. For such a model it is intractable to adjust the parameters to optimize the ability of generating the exact probability distribution of the observed set of patterns. A possible solution is to limit this combinatorial explosion by maximizing a lower bound on the probability of the observations.

To see how this is possible, a comparison can be made between the possible explanations of a pattern and the alternative configurations of a physical system. Each possible set of causes is called an *explanation* of a pattern. The log probability that a model with parameters θ, generates a particular pattern u, is the log of the sum of probabilities that a certain explanation α, is chosen by the model and that this particular explanation generates u:

$$\log p(u|\theta) = \log\left(\sum_{\alpha} p(\alpha|\theta)p(u|\alpha,\theta)\right) \tag{4.1}$$

The posterior probability of an explanation α, given the pattern u, and the model parameters θ, is the probability that α was chosen by the model and that it generated u, divided by the sum of probabilities that another explanation α', was chosen and generated u:

$$P_\alpha(u,\theta) = p(\alpha|u,\theta) = \frac{p(\alpha|\theta)p(u|\alpha,\theta)}{\sum_{\alpha'} p(\alpha'|\theta)p(u|\alpha',\theta)} \tag{4.2}$$

In statistical physics the energy of an explanation, E_α, can be written as the negative log of the posterior probability that α was selected from a model with parameters θ, given u:

$$E_\alpha(u,\theta) = -\log p(\alpha|\theta)p(u|\alpha,\theta) \tag{4.3}$$

The posterior probability of an explanation as given in equation 4.2 is in statistical physics related to the energy of a system at equilibrium given by the Boltzmann distribution (with a temperature of 1):

$$P_\alpha(u,\theta) = \frac{e^{-E_\alpha}}{\sum_{\alpha'} e^{-E_{\alpha'}}}, \tag{4.4}$$

where indices u and θ have been omitted for clarity. When there are more alternative explanations, it is possible to design a representation that takes advantage of the entropy across alternative representations (Zemel and Hinton, 1994). Using equations 4.3 and 4.4, equation 4.1 can now be rewritten in terms of the difference between the expected energy of an explanation and the entropy across explanations, which in statistical physics is called the Helmholtz free energy:

$$\log p(u|\theta) = -\left[\sum_\alpha P_\alpha E_\alpha - \left(-\sum_\alpha P_\alpha \log P_\alpha\right)\right], \tag{4.5}$$

where the second term is the entropy of the distribution assigned to the various alternative representations.

Any probability distribution over explanations has at least as high free energy as the equilibrium, or Boltzmann distribution (Thompson, 1988) as given in equation 4.4. The trick is now to rewrite equation 4.5 using a more convenient probability distribution Q, instead of the true posterior probability distribution P, and make up for the difference between Q and P by subtracting the Kullback-Leibler divergence (Kullback, 1959) between Q and the true posterior probability distribution P:

$$\begin{aligned}\log p(u|\theta) &= -\left[\sum_\alpha Q_\alpha E_\alpha - \left(-\sum_\alpha Q_\alpha \log Q_\alpha\right) \quad - \quad \sum_\alpha Q_\alpha \log\left(\frac{Q_\alpha}{P_\alpha}\right)\right] \\ &= -\left[F(u,\theta,Q) \quad - \quad \sum_\alpha Q_\alpha \log\left(\frac{Q_\alpha}{P_\alpha}\right)\right]\end{aligned} \tag{4.6}$$

The non-equilibrium posterior probability distribution Q, always has at least as high free energy as the energy at equilibrium. The last term in equation 4.6, the Kullback-Leibler divergence, cannot be negative (Kullback, 1959) so ignoring it results in a lower bound on the log probability of the data given the model. The Helmholtz free energy $F(u,\theta,Q)$, is now based on the non-equilibrium posterior probability distribution Q.

4.3 Basic principles of the Helmholtz machine

Not only did Helmholtz provide us with the math that enabled the formal description of the models described in the previous section, he was also proponent of the idea that the perceptual system uses generative models. Therefore the implementation of the principles discussed in the previous sections in a neural network model, was named after him. A Helmholtz machine (Dayan et al., 1995) applies the principles discussed in the previous section to infer the probability distribution Q, over the underlying causes from presented patterns in a recognition model. The recognition model is implemented as a separate model with its own parameters ϕ. Furthermore the Helmholtz machine has a generative model with parameters θ, which is used to train the recognition model. During training, the parameters of both models are optimized to maximize the lower bound of the log probability of the data, by maximizing $-F(u,\theta,Q(\phi))$.

The Helmholtz machine as depicted in figure 4.2 is a connectionist system with multiple layers of binary stochastic units. Let $\mathcal{G}$ be a Helmholtz machine in which the units are connected by two sets of connections with adjustable weights to implement separate recognition and generative models. The external inputs to the network as well as the outputs of the units are denoted by the symbol y. H is the set containing indices i, such that y_i is the output of a unit, I is the set containing indices i, for which y_i is an external input to the network, and K is the set containing indices i, for which y_i is the output of a unit a hidden layer of the network other than the topmost hidden layer. Let ϕ be the set of weights that implement the recognition model with a unique value w^r_{ij}, for each weight between the input of unit i and the output of unit j of every connected pair of units in the recognition model, and let θ be the set of weights that implement the generative model with a unique value w^g_{ij}, for each weight between the input of unit i and the output of unit j of every connected pair of units in the generative model. To provide for a bias, each unit is connected to a single bias unit, which always has a value of 1. B is the set of bias units in the generative model that are connected to the units in the top layer of the network. Both sets of connections connect only units between different layers, so the state of a unit in layer l is independent of the states of the other units in this layer. This means that the distributions of both the recognition and the generative model are factorial in each layer.

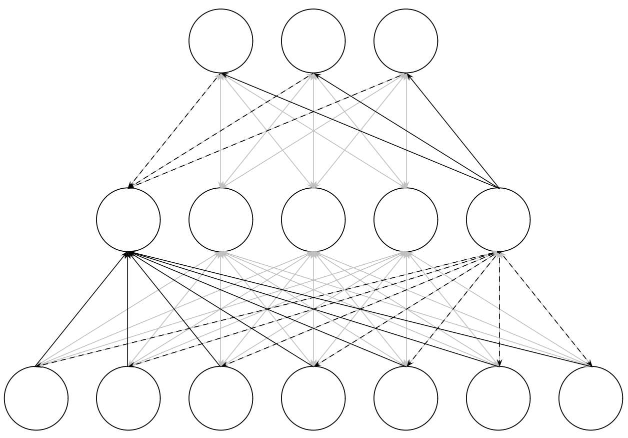


Figure 4.2: The Helmholtz machine. The solid lines represent the bottom-up recognition model, and the dashed lines the top-down generative model. Some lines are depicted in gray for clarity.

The recognition model is used to produce the non-equilibrium posterior probability distribution Q, from equation 4.6. Given an input pattern the recognition model assigns a state to every unit in the network, using the model's parameters to produce an explanation of the input pattern. For a certain input pattern, the probability q_i, that the state y_i, of the i^{th} unit for $i \in H$, is 1, is computed from the bottom to the top layer using the recognition weights $w^r_{ij} \in \phi$, according to:

$$q_i = f_i(net_i), \tag{4.7}$$

where net_i is the net input to unit i:

$$net_i = \sum_{j \in K \cup I} w^r_{ij} y_j \tag{4.8}$$

The function f_i, can be any differentiable function, but is usually chosen to be the sigmoid function from equation 2.4. The activity or state y_i, of unit i is 1 with probability q_i (or 0 with probability $1 - q_i$).

An explanation α, is the complete assignment of all states $y_i \in H$. The contribution to the probability of choosing α of each unit is q_i for units that are on ($y_i = 1$), and $1 - q_i$ for units that are off ($y_i = 0$). The total probability of choosing α from the recognition model is the product of all contributions of all units other than those in the input layer. In this way it is possible to compute the probability of α, given the model parameters ϕ, and an input pattern u:

$$Q_\alpha(u,\phi) = \prod_{i\in H} (q_i)^{y_i}(1-q_i)^{1-y_i} \tag{4.9}$$

The Helmholtz free energy can be computed using the generative model through $E_\alpha(u,\theta)$ from equation 4.3. To determine the energy of an explanation, the generative probabilities that explanation α reproduces the states of the hidden units and the input units are computed, using the generative model with parameters θ. The top-down connections $w^g_{ij} \in \theta$, are now used to determine the probability p_i, that the states in layer $l+1$ generate the states in the underlying layer l:

$$p_i = f_i(net_i), \tag{4.10}$$

where net_i is the net input to unit i:

$$net_i = \sum_{j\in H\cup B} w^g_{ij} y_j \tag{4.11}$$

Because in the generative model there is no input to the units in the top layer, the probabilities in the top layer are computed using only the inputs from the bias units that are connected to the units in the top layer. Furthermore, function f_i is again the sigmoid function from equation 2.4.

The contribution of the generative probability p_i, of unit i of reproducing α is p_i for units that are on ($y_i = 1$), and $1-p_i$ for units that are off ($y_i = 0$). The total probability of generating α from the generative model is the product of all contributions of all units other than those in the input layer:

$$p(\alpha|\theta) = \prod_{i\in H} (p_i)^{y_i}(1-p_i)^{1-y_i}, \tag{4.12}$$

The probability p_i, that the states in the second layer ($l = 2$) generate the states in the input layer ($l = 1$), can also be determined with equation 4.10, resulting in the probability of the input pattern given explanation α, and generative model parameters θ:

$$p(u|\alpha,\theta) = \prod_{i\in I} (p_i)^{y_i}(1-p_i)^{1-y_i}, \tag{4.13}$$

The energy of α from equation 4.3 can now be determined using equations 4.12 and 4.13:

$$\begin{aligned} E_\alpha(u,\theta) &= -\log p(\alpha|\theta)p(u|\alpha,\theta) \\ &= -\sum_{i\in H\cup I} y_i \log p_i + (1-y_i)\log(1-p_i) \end{aligned} \tag{4.14}$$

An unbiased estimate of $F(u,\theta,\phi)$ can be determined based on an explanation α, drawn from Q_α as given in equation 4.9 and the energy E_α, as given in equation 4.14:

$$\begin{aligned} F_\alpha(u,\theta,\phi) &= E_\alpha(u,\theta) + \log Q_\alpha(u,\phi) \\ &= \sum_{i\in H\cup I} y_i \log \frac{q_i}{p_i} + (1-y_i)\log\left(\frac{1-q_i}{1-p_i}\right) \end{aligned} \tag{4.15}$$

The total negative free energy across all patterns can simply be found by summing over the proportion of each pattern to the probability distribution Q, which is produced by the recognition model ϕ:

$$-F(\theta,\phi) = -\sum_u F(u,\theta,\phi) \tag{4.16}$$

Training the weights now involves a stochastic gradient ascent in the total negative free energy from equation 4.16.

4.4 The wake-sleep algorithm

Training the recognition and the generative model involves adjusting the weights in the direction of the proportion of each weight to the derivative of the free energy across the provided input patterns. These derivatives are quite complex because they have to take into account the effects that changing the states of the units in one layer has on the states of the units in other layers. A solution to this involves an algorithm that trains the network in two separate phases, called the wake-sleep algorithm (Hinton et al., 1995).

Input patterns are presented to the bottom layer of the network during the wake phase, and the states of the units are computed bottom-up, using the recognition model. The top-down generative weights are then adjusted to minimize the Kullback-Leibler divergence between the actual states and the generated probabilities that the states in layer $l+1$ generate the states in layer l. During the sleep phase the states of the units in the network are computed top-down using the generative model, thus generating a random instance from the generative model. The true underlying causes of the generated pattern are known now, so the recognition model can be trained using the target values of the hidden units.

Training the models can be compared to a communication between a sender and a receiver, in which the sender communicates the representation of an input pattern and the difference between the input pattern and its reconstruction from the representation. Assuming the receiver knows the parameters of the selected model, it can reconstruct the input pattern. The goal of learning is to minimize the length of the communication needed to send the input patterns in this way. Minimizing the length of the communication forces the model to learn economical representations that allow the input patterns to be reconstructed accurately.

When a pattern u, is presented to the network, the state y_i, of each unit i, is 1 with probability q_i, which is computed from equation 4.7. The values of q_i and y_i are computed in each layer, from the bottom to the top. The collective states of all units, other than those in the input layer, is the total representation α. The states of the units in each layer other than the input layer are now communicated to the layer below, using the distribution p_i for units that are on and $1-p_i$ for units that are off, where p_i is computed according to equation 4.10. Shannon's coding theorems states that it requires $-\log r$ bits to send a state with probability r under a distribution known to both the sender and the receiver (Casti, 2000). Communicating the state y_i, of unit i, using the distribution $(p_i, 1-p_i)$ requires:

$$C(y_i) = -y_i \log p_i - (1-y_i)\log(1-p_i) \tag{4.17}$$

bits, where $C(y_i)$ is called the *description length* of unit i. The description length of the total representation is the cost of describing the states of all units other than those in the input layer in this way. Furthermore, the difference between the input pattern and its reconstruction from the representation needs to be specified, which is the cost of describing the input pattern given the total representation. The total description length of communicating the patterns is this way is:

$$\begin{aligned} C(\alpha, u) &= C(\alpha) + C(u|\alpha) \\ &= \sum_{i \in H} C(y_i) + \sum_{i \in I} C(y_i|\alpha) \end{aligned}, \tag{4.18}$$

where H ranges over all units other than those in the input layer, and I ranges over all units in the input layer. Because the network consists of stochastic units, the representation of an input pattern is not always the same, so the recognition weights determine a conditional probability distribution over the total representations. The recognition weights are not changed during the wake phase, so it is possible to adjust the generative weights to minimize the expected cost $\sum_\alpha Q(\alpha|u)C(\alpha, u)$, of describing the input pattern with the chosen representation. The generative weights are adjusted in the direction of the proportion of each weight to the derivative of equation 4.18, using the delta rule:

$$\Delta w_{ij}^g = \gamma y_j (y_i - p_i), \tag{4.19}$$

where γ is the positive learning rate parameter. This makes each layer in the network other than the input layer better at reconstructing the states in the layer below.

In the next phase, the sleep phase, the recognition model is trained by generating a random instance from the generative model that is computed by activating the units top-down according to equation 4.10. Because the network consists of stochastic units, this process will generate different 'fantasy' patterns, providing unbiased samples from the generative model. The probabilities that the states of the units in layer $l+1$ would be represented by the recognition model from the states of the units in layer l, are computed with equation 4.7. Again the delta rule is used to compute the weight updates of the recognition model:

$$\Delta w_{ij}^{r} = \gamma y_j (y_i - q_i), \tag{4.20}$$

where γ is the positive learning rate parameter.

From the previous section it is clear that it is possible to design a representation that takes advantage of the entropy across alternative representations, if there are many alternative ways to represent an input pattern. In terms of description length, the cost of describing an input pattern is then:

$$C(u) = \sum_{\alpha} Q(\alpha|u) C(\alpha, u) - \left(- \sum_{\alpha} Q(\alpha|u) \log Q(\alpha|u) \right), \tag{4.21}$$

where the second term is the entropy of the distribution that the recognition weights assign to alternative representations. So training the recognition model does not involve adjusting the weights to maximize the probability of choosing the representation that minimizes $C(\alpha, u)$, but to make the distribution Q as similar as possible to the Boltzmann distribution from equation 4.4, which in terms of minimum description length is:

$$P_\alpha(u, \theta) = \frac{e^{-C(\alpha,u)}}{\sum_{\alpha'} e^{-C_(\alpha',u)}}, \tag{4.22}$$

$C(u)$ is minimized when the probabilities of the alternative representations are exponentially related to their costs by the Boltzmann distribution, which is the posterior distribution over representations given the input patterns and the generative model. This can be accomplished by updating the weights in an iterative fashion according to equations 4.19 and 4.20.

The sleep phase trains the recognition model from patterns that are produced by the generative model rather than the real input patterns. Furthermore, distribution Q produced by the recognition weights, is a factorial distribution because the states of the units in layer l are conditionally independent given the states in the layer $l-1$ below. This makes it impossible for Q to exactly match the posterior distribution from equation 4.22. The reason this method still works, is in equation 4.6. During the wake phase, the generative model is trained to make the distribution P match Q, limiting the loss caused by the factorial distribution of Q. The Kullback-Leibler divergence between Q en P, which is the cost of describing an input pattern using Q exceeds that of P, is minimized by training the generative model.

Learning with the wake-sleep algorithm as described here is generally performed in an unsupervised fashion. Although it is possible to train a Helmholtz machine with supervised learning using certain units as output units (Dayan and Hinton, 1996), assigning certain concepts to the activity of specific units as described in the previous chapter is something that is to be avoided here. As opposed to training algorithms used for backpropagation networks, the minimum description length principle is the basis of the learning methods used in the Helmholtz machine. There are basically two ways in which this principle can be applied to learn a classification task.

First, a Helmholtz machine can learn to represent a probability distribution over the input data. After presenting a Helmholtz machine with a sufficient amount of training data, the machine has built an internal model of the input patterns provided during training, and can reproduce the input patterns with the same probability as in the training set. This can be achieved by activating the units in the topmost layer with a probability drawn from their generative bias values, and computing the activities of the units in the lower layers using the generative model. When performed repeatedly, this process will ultimately generate each input pattern with the same distribution as in the training set. This method is of course very useful if one wants to build a model that can reproduce input patterns with a certain probability distribution, but this method cannot be used for a classification task.

The second method is more suitable for classification. Although the wake-sleep algorithm itself is a unsupervised training method that involves a local update rule, it is possible to use a Helmholtz machine for a supervised classification task. Training in this fashion involves a supervised learning method, in which a separate Helmholtz machine is taken for each class. Input patterns are only provided to that Helmholtz machine that corresponds to the class the pattern belongs to. The minimum description length principle can now be used as measure for how well a certain concept can be represented by a Helmholtz machine. To test how well this method performs on novel inputs, each pattern is presented to all Helmholtz machines, and the machine that can represent the pattern using the fewest amount of information is said to represent the class of that input pattern. So the Helmholtz machine that has learned to represent an input pattern with the shortest description length, has the best internal representation of the class that pattern belongs to.

The Helmholtz machine, as described in this chapter is an implementation of Mumford's proposals for mapping the approach of generative models onto the brain. The wake-sleep algorithm implements the minimum description length principle, which is used to build models of the processes that are going on in the outside world. Although Mumford's pattern theory is mostly hypothetical concerning actual brain processes, it is not unlikely that the Helmholtz machine models certain aspects of the human brain closely.

Chapter 5

Music

5.1 Introduction to terminology

Music has left its traces in human culture and society during the course of history, and still is an important aspect of human life nowadays. Already in the days of the Greek philosophers, Plato saw how much music could influence people and concluded that the spiritual temperature of an individual or society could be marked by the music they performed or listened to. What the exact origins of music are and how mankind got involved in making music is still largely unknown. Some argue that the structure in the music of animals and humans shows so much similarity that it is likely that the roots of music lie closer to the brain structure of our primaeval non-human ancestors than to the more recently evolved reasoning cortex (Gray et al., 2001). Whether it is the case that humans can make an exclusive claim to music, or that music predates humans, in any case it is clear that music is universal among all humans. In all cultures around the world people practise the art of making music, and enjoy listening to their musical creations. Most people seem to have the innate ability to memorize, produce and recognize musical patterns which makes it possible to pass on tunes between individual persons and from generation to generation. Before we explain how this extraordinary ability is achieved by the human brain, we will first shed some light on the nature of music and introduce along the way some terminology that will be used later on.

The basic component of music is sound. Sound is a vibration in the air that is caused by a moving or oscillating body, such as a string or a speaker. The amount of oscillations per time unit is called the *frequency*, which is often measured as the number of vibrations per second in Hertz (Hz) and the *amplitude* is a measure for the magnitude of the oscillation. In more simple terms: the amplitude determines the loudness of a sound, and the frequency its pitch. Vibrating bodies don't necessarily vibrate at one frequency; they often produce vibrations at several different frequencies. As such a body vibrates, it causes the air around it to vibrate and the vibrations are propagated through the air as sound waves, which can be picked up by the human ear.

Most musical instruments have the property that they can produce one or more discernible tones. As vibrating bodies often produce more than one frequency, each tone is made of a number of distinct frequencies, the lowest of which is called the fundamental and produces the *pitch* of that tone. The other frequencies that are produced by the instrument are often approximately integer multiples of the fundamental frequency, called harmonics. These overtones and the balance of their amplitudes are important factors for giving each instrument its characteristic timbre.

A musical instrument or the human voice can produce a tone with a certain duration at a fixed pitch. Such an instance of a tone is called a *note*. The duration of the onsets between notes that are played successively is what produces a sense of *rhythm*. Rhythm is a very important aspect of music because it creates a sense of structure in time. Not only does sounding different notes after each other produce rhythm, the combination of the relations between pitches and durations of the successive notes produces *melody*. In addition to playing different notes subsequently, it is also possible to play two or more different notes at the same time. The relation between notes that are sounded simultaneously is what constitutes *harmony*.

The distance or relation between the pitches of successive or simultaneously sounding notes, is called the *interval*. The frequency ratio between two pitches determines how well the combination of the two sounds. In many cultures the basic intervals in harmony are those using the ratios with the lowest integers, such as 1:1, 1:2, 2:3, etcetera. For example, a certain pitch with a frequency of 440 Hz and another one with a frequency of 880 Hz, have a frequency ratio of 1:2. Putting different notes together, using the rhythmic, melodic and harmonic aspects of the combination of notes results in a musical piece, which often consists of a lot of individual and discernible units. Each different note can be seen as a unit, but also groups of notes that have certain properties in common, such as the intervals between successive or simultaneously sounded notes, or the duration and similar rhythmic properties of the notes. Repetition and symmetry of groups of notes that have certain aspects in common is what makes the units, phrases and above all, the organization and structure in music.

In each period in history, and in each culture composers use a particular combination of units and rules to create a song or a musical piece. For instance, Western music is characterized by the use of a 12-note scale and a trend toward complexity in harmony, rhythm, and orchestration. Western music in the early Middle Ages was mostly monophonic, which means that it consisted of just one part. In later periods other parts were added, which were often subservient to the melody. Polyphony emerged in the Renaissance period and culminated during the Baroque period in the musical technique that is known as counterpoint. Counterpoint aims at creating several independent melodic voices, as opposed to monophonic music, which has just one part and homophonic music, which consists of a dominant melody part that is accompanied by other parts. The melodic interaction between the different voices rather than the harmonic effects generated by the intervals between the voices is important in counterpoint. Furthermore, the rules that govern the melodic interactions are often very strict. Towards the end of the Baroque counterpoint lost its dominant role, as harmony became the predominant organizing principle in music. As described before, harmony is the relation between the pitches of two or more notes that are played at the same time. The combination of three or more different pitches that are sounded simultaneously, or nearly simultaneously over a period of time is called a chord. Generally speaking, any combination of three or more pitches is a chord, albeit that in Western music some combinations are more prominent than others. Since the late Baroque, chords and patterns of chord progressions are very important in Western music; these are still among the main aspects of popular and jazz music.

Within the class of Western music there exists a great variety of musical styles which are achieved by very complex conventions and rules. Each period, style and composer has its own characteristics, but there are some properties that all Western music have in common. Rhythm is often present in Western music as a periodic structure of stressed and unstressed notes or beats. When this structure is regular, the music is characterized by a repeating basic rhythm which can be perceived as a regular pulse. The presence of repeating rhythmic patterns with a regular structure is called meter. Tempo is a related concept that determines how fast a piece is played. The length of notes and the time between the onsets of different notes is what makes rhythm, but timing and length are always relative to tempo. Musical pieces can be performed at different tempos, moreover the tempo can change abruptly, slow down or speed up within a musical piece, but this does not affect the mutual proportions of rhythmic patterns. Together with notes and relations between the pitches of notes, duration and temporal structure are the most important features of Western music.

5.2 Music perception

The process of music perception starts at the ear, which is an ingenious device that picks up sound waves and transforms them into neural pulses that are transported deeper into the brain. Inside the ear are several thousands sensory cells that look like tiny hair cells, that resonate at specific frequencies. Each hair cell represents a small frequency band and produces an amount of neural activity that is proportional to the amplitude of the frequency at which it resonates.

The hair cells in the ear as well as the deeper lying cells in the auditory cortex are arranged in a certain order such that similar frequencies are represented by neighboring cells. Due to the physical structure of the ear, the processing of sound already begins in the inner ear which sorts complex sounds, for instance a tone produced by a musical instrument, into its constituent elementary frequencies.

As described before, musical instruments as well as the human voice produce a whole range of sounds at different frequencies that are mostly harmonic overtones. The different frequencies of which a sound consists that are picked up by the hair cells inside the ear, are however not perceived individually by humans. Instead of hearing the whole spectrum of the fundamental frequency and the overtones, humans perceive the fundamental as the pitch of a sound. This aspect of hearing is very important in music. For instance it makes it possible for two sounds that might consist of different frequencies, such as the same note played on a piano and a violin or sung by different persons, to be perceived as the same pitch. Pitch perception does not take place at the level of the ear, because the different frequencies of which a sound consists are transported separately to the brain, but the brain actively constructs pitch perception. Research has shown that the reconstruction of pitch is a very important aspect of the processing of sound in the brain. Even when in an experimental setup the fundamental frequency of a tone is deleted while keeping the harmonics, the missing fundamental is still perceived as the pitch of that sound (Whitfield, 1980). This implies that the information about individual frequencies is somehow combined to construct pitch. Not only humans have the ability to perceive pitch instead of the individual frequencies of which each tone consists, this property has also been found in the brain of animals, such as birds, bats, cats and monkeys (Weinberger, 1994a). The organization of the hair cells in the inner ear such that neighboring cells are tuned to adjacent frequencies, is also maintained in the auditory cortex. An important difference however, is that this organization is not at the level of individual frequencies, but at the level of *pitch* (Pantev et al., 1989, 1991). Certain areas in the auditory cortex respond in essentially the same way to a variety of sounds that all have the same fundamental but do not necessary share any frequencies. The neurons in these areas are very important for the ability to perceive pitch, and have a spatial order according to this abstract property, such that adjacent areas respond to adjacent pitch, just as the order of the keys on a piano. The obvious representation of pitch in the neural structure of the brain implies that it is an important concept in auditory perception.

Although very important in music, pitch perception is not the only property that accounts for the phenomenon music. The same mechanisms that are responsible for producing a sense of pitch are also involved in another important aspect of music: harmony. As described before, the ratio of the frequencies of tones that are sounded together is very important in harmony. However, the human brain is not a computer that can easily multiply frequencies and compare the results with each other to produce a sense of harmony. Just as in pitch perception, the overtones seem to play a crucial role in harmony perception. It was Helmholtz who found that sounds which are harmonically related, have overtones in common (Helmholtz, 1877). The neural basis for this is less complicated than would be required for computing multiplications of frequencies. As the human ear is constructed in such a way that it activates a specific neuron for each frequency band, partly the same neurons will be activated for sounds which have overtones in common. Because these neurons have a certain amount of overlap, the overtones don't have to be exactly the same, as Helmholtz stated, but can diverge within the range of that frequency band. The number of coincidence neurons which respond to two different harmonic frequencies could be used to determine their harmonic relation, but it is debatable if such a simple mechanism can account for all different aspects of harmony (Fishman et al., 2001). Due to the structure of the auditory system other related mechanisms are sometimes activated by harmonic relations. For example, frequencies with approximately simple integer ratios produce sequences of action potentials that show stronger temporal correlations than frequencies with ratios of high integer numbers. Of course the frequencies are not always produced in the same phase, but due to the stochastic nature of neuronal representations it is still possible to detect synchrony in firing patterns. In addition, there is a mechanism that activates neurons that correspond to the difference in frequency between two overtones of different frequencies. This means that the difference between two frequencies can sometimes be perceived as a tone that is actually not present. Harmony is also directly related to the emotional aspects of music. Certain ratios between two different frequencies result in the activation of a part of the brain that is responsible for pleasant feelings, while other specific ratios result in the activation of areas that are involved in other emotions (Blood and Zatorre, 2001).

The previous section has pointed out that although music comes to us as an immediate and wholistic experience, it consists of clearly discernable units. Not only does the auditory cortex process pitch rather than raw sound frequencies, brain cells have been found that process more general temporal aspects of music such as specific melodic patterns and rhythm. Structuring the auditorial input is an important function of the brain, moreover phenomena like music and language seem to based on this property. Rhythm is an important feature of music that is recognized by grouping sound sequences into discrete phrases. This process which in other cognitive studies is referred to as *chunking*, does not only enhance the auditory memory, but also makes it easier to recognize the same pattern independent of the speed at which it is played (Weinberger, 1994b). Rhythm is however not the only aspect that is produced by the structuring process, there also is a different mechanism in the brain that is responsible for structuring successive pitches. This means that individual pitches are not processed independently, but that the place and relation of a pitch with respect to other pitches is important. For instance, certain cells in the auditory cortex react differently to the same pitch when it is part of a sequence of ascending pitches than when it is part of a sequence of decreasing pitches (Zatorre, 2005). Together, all these different units in music that are the result of the structuring performed by the brain, constitute a set of rules that govern the proper combination of elements. For instance, in Western music there is a strong expectancy for specific periodic rhythmic patterns as well as certain chord progressions. The presence of neural structures that are specifically dedicated to music and the amount of brain regions that participate in the different aspects of music perception as well as emotional reactions, shows that music is not a fancy by-product of the complex human brain, but that it has a strong biological basis.

5.3 Music classification

People which are interested in music often like or dislike certain styles and can easily recognize the style or composer of a certain musical piece, even if they hadn't heard it before. To assist people in the task of recognizing music, computer programs have been developed that use different features of music to assign labels to musical pieces, such as the composer or a certain musical style. Music classification has become a growing field of interest in music technology, as many people have access to large collections of digital musical pieces. Databases of music files are often organized in directory structures classified by genre or artist, but as these classification methods are not always accurate and because of the problems that arise due to language differences, a system that makes classifications based on musical content would be much more appropriate. Such a system could also be used to automatically classify new music using the existing classification system specified by the user. Although music classification using algorithmic methods is a relatively new field, extensive research has been performed on the nature of musical units, and what rules are used to combine these units into musical pieces.

Digital music files can roughly be divided in two distinct classes: files in wave format, and files that consist of notes. Wave format files are direct recordings of sound waves through a microphone or similar device, transformed to a digital signal and stored on some digital medium. Methods from digital signal processing theory are often applied to extract features from the spectral analysis of frequency bands of the digital wave files. Frequency bands are computed using bandpass filters that split the incoming signal consisting of multiple frequencies into separate frequencies that can be processed independently. This process is the equivalent of frequency separation of the human auditory system. Several other methods such as the extraction of power spectral density, smoothness in frequency domain and the development of wave energies, are used to extract some more low-level properties of the musical piece (McKinney and Breebaart, 2003). These methods extract more abstract representations of basic musical properties such as meter, tempo, rhythm and the timbre of the musical instruments or the human voice.

The other class of music files does not contain digital information of sound waves produced by musical instruments or voices, but contains information on the pitch, onset, duration and volume of individual notes. For example, it contains information on when to press and release

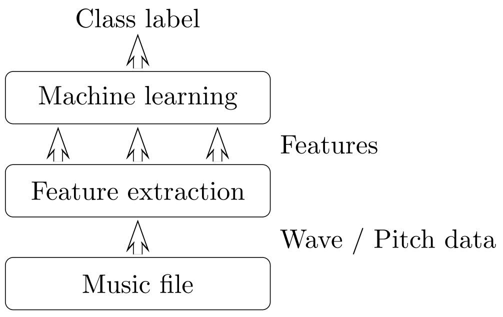


Figure 5.1: Schematic overview of music classification

a specific key on a piano, and with what velocity, but the actual sound that is produced by the piano is not stored in the file. A commonly used format for storing such files on digital computers is the MIDI file format. This representation of music can be used to extract high-level information on musical features, rather than sound features. As opposed to wave format files, this format does not require digital signal processing theory for feature extraction, but the individual notes can be studied with methods that are directly derived from music theory (McKay and Fujinaga, 2004; Li and Sleep, 2004). Melodic features can be extracted from the intervals between subsequent notes, upward or downward moving sequences of notes, and modulations between different keys. The intervals between pitches of simultaneously played notes, and chord progressions are used for information on harmony. Meter, note onset and note length can be used to extract rhythmic features, and the change in volume of subsequent notes for information on dynamical features.

Feature extraction of digitized music files and the representation of these features is often limited to short fragments. Because of the nature of the methods and representation of the features that are used to extract information from music files, these methods are often applied to subsequent and possibly overlapping parts of the music file. For instance, it is not of much use to know that the average interval between the notes of a melody has a certain value, while the intervals between subsequent notes that are much more interesting, take place on a smaller timescale. Therefore a musical file is divided into several time-frames, and features are computed for each frame. The computed feature snapshots of small parts of music can then be used for further analysis.

Whereas statistical analysis methods used to be the standard in music classification, machine learning methods are becoming increasingly popular. Such methods include hidden Markov models, PCA analysis, neural networks and support vector machines (Basili et al., 2004; Harford, 2003; Maddage et al., 2003). The main goal of machine learning methods is to learn to classify musical pieces by genre or composer, using sets of features such as those described before. This process is illustrated in figure 5.1. Most machine learning methods are limited in the sense that they can only process snapshots of situations, and cannot learn any temporal dependencies between those snapshots. Apart from the fact that all this feature extracting is not always biologically plausible, these methods can in most cases not be used to learn temporal dependencies between different snapshots, and therefore they rely on the temporal features that are extracted from the music. Furthermore, feature extraction as described before is because of the nature of the methods used also limited to short fragments of music. It is common practice in the field of music classification to cut a musical piece into small parts. These parts are preprocessed by feature extraction methods, and the features are provided to machine learning methods as snapshots. The liquid state machine in combination with the Helmholtz machine provides a framework in which both the problem of extensive preprocessing and the limitation of using snapshots of small parts of the data disappear.

Chapter 6

Methods and implementation

6.1 Music classification as learning task

Investigating the questions in section 1.3 into more depth involves an appropriate learning task for testing the different aspects of the application of a Helmholtz machine as readout mechanism of a liquid state machine. Such a learning task should of course involve time series data in which the temporal dependencies play an important role to justify the use of a temporal liquid filter. Furthermore, the Helmholtz machine can only be used for unsupervised learning or for classification tasks as described at the end of chapter 4, so the learning task should be a classification task that can only be solved by finding temporal dependencies between different input patterns in time series data. Such a task should of course not be too easy, because it is difficult to conclude anything from a learning algorithm that always solves the task regardless of the parameter settings. Because the artificial neural networks used in this project are not mere theoretical inventions, but try to model certain aspects of the human brain, the learning task should preferably be of a kind that can also be solved by the human brain. A task that satisfies these requirements and does not only involve quite complex brain mechanisms, but is also very easily solved by humans, is the classification of music. A machine learning algorithm that doesn't rely on preprocessing can only solve this task by finding musical properties such as rhythm and melody, that are inferred from temporal dependencies in the musical data, such as the relation between the lengths and pitches of successive notes. However, in order to create a music classification task that depends mostly on temporal dependencies between the different notes, there should not be too much difference in harmonic features between the different classes. Harmony can be derived from the proportion between the pitches of different notes that are sounded at the same time. That means that harmony can be derived from single input patterns in time series data of music. If the classes that have to be learned have very distinct harmonic properties, the task can easily be solved without finding temporal dependencies.

As described in the previous chapter, most other methods used for music classification heavily rely on extracting all sorts of features. The feature extraction algorithms used in these methods are often applied to find temporal relations in the data, in order to avoid the inference of temporal relations in the machine learning methods that are used for further processing of the features. This is however not the way the task is solved in this project. The Helmholtz machine is based on extracting features from input data, and builds a generative model of the processes that caused the data to occur. Too much preprocessing of the data complicates this task. Besides, we don't want to test the performance of feature extraction algorithms to find temporal dependencies in the data, but the capability of a spiking neural network as liquid filter to extract those temporal dependencies.

A music classification task involves finding the corresponding class for a certain musical piece, based on its content. A set of musical pieces can be divided into different classes based on several criteria. The criterion used in the music classification task for this project is the composer. Among the most famous composers in Western classical music are without doubt Johann Sebastian Bach and Ludwig van Beethoven. Both of them have written an extensive opus, including not only orchestral and chamber music, but also vocal, choral and keyboard works. Although not of the same period, the works of Bach and Beethoven have much in common. Beethoven played and studied the work of Bach and has most certainly been influenced by this in his compositions. Two sets of musical pieces that are not only very representative for the two composers, but are also intended for the same type of instrument, are: the 'Well-Tempered Clavier' by Bach, and the 'Piano Sonatas' by Beethoven.

The Well-Tempered Clavier is a major work of music for keyboard instruments, such as the harpsichord or the piano. It consists of two books with twenty-four pairs of preludes and fugues each. The first pair is in C-major, the second in C-minor, the third in Csharp-major, the fourth in Csharp-minor, etcetera. Every major and minor key is represented in each of the two books in ascending order. The Well-Tempered Clavier is a work of great depth and diversity, and has provided the harmonic basis for classical music from Bach onwards. Just as the Well-Tempered Clavier is a work of major importance in Western classical music, the 32 Piano Sonatas written by Beethoven are among the most prominent in their class. A piano sonata is a musical piece written for unaccompanied piano. Sonatas are usually written in several movements, mostly three or four, and satisfy certain rules, for example the modulation to certain keys in different movements. Because these two sets are very important in the history of Western music, and because their harmonic properties are not too different as well, they are used as the two classes in the music classification task.

6.2 Setup

To compare the performance of the Helmholtz machine with other readout mechanisms, another neural network is required that has certain aspects in common. For example: most neural networks can process real numbers within a certain interval, whereas the Helmholtz machine as implemented in this project can only process input patterns that consist of ones and zeros. For a fair comparison a neural network is needed that has no difficulty learning from input patterns that consist of ones and zeros, and has a comparable amount of complexity in terms of non-linear functions and the number of adjustable weights. Such a neural network is the multilayer perceptron as described in section 2.3. The multilayer perceptron can solve non-linearly separable classification tasks, and is at least as good as most methods used by others, such as linear perceptrons. The structure of the multilayer perceptron is not much different from the recognition and generative models of the Helmholtz machine. Both networks use the same summation and squashing functions, however the learning algorithms of both networks are based on different principles. Because of this difference not much can be explained about the nature of the possible difference in performance between the multilayer perceptron and the Helmholtz machine. However, to compare the performance of a Helmholtz machine as readout mechanism of a liquid state machine with other readout mechanisms, a neural network that can solve quite complex problems serves as a good reference.

Neural networks are not just designed and tested; they first have to be trained. During training the network learns to extract features from the input patterns in the case of a multilayer perceptron, or learns to build a model of those patterns in the case of the Helmholtz machine. The data set that is presented to the network during training is called the *training set*. When the learning phase is finished, the performance of the network is tested on input patterns that were not presented to the network during training, but belong to the same classes. This novel data set is called the *test set*. In this way the ability of the network to learn the characteristics of a certain class and generalize this to data it has never seen before, is tested. Because the multilayer perceptron and the Helmholtz machine operate on a different basis, a performance measure is needed that can be used to compare the performance of both algorithms with each other.

Multilayer perceptrons can be used for classification tasks by assigning an output unit for each class. The activity of an output unit can be used as measure for the correspondence of an input pattern to a certain class. Therefore in the multilayer perceptron used in the experiments in this project the target values for the output units are 1 for the unit that represents the correct class, and 0 for the other units. In this way, the network learns to assign a high value to the unit that represents the class of an input pattern, and a low value for all other output units. Because a certain input pattern can cause an activity other than zero in more than one output unit, the unit with the *highest* activity is said to correspond to the class the network assigns to that input pattern.

Unlike the supervised backpropagation algorithm used for the multilayer perceptron, the Helmholtz machine uses the unsupervised wake-sleep algorithm for training. To compare the capabilities of a Helmholtz machine with other algorithms, a supervised learning method was devised in which the performance of the Helmholtz machine to build models of the input data is tested, while the training algorithm itself remains unsupervised. The minimum description length principle is used as a performance measure in the experiments in this project, by implementing a separate Helmholtz machine for each class. During training the input patterns are presented to the single Helmholtz machine that represents a class, and the wake-sleep algorithm computes the updates for the weights of that Helmholtz machine to learn a better representation of the input pattern in terms of description length. When the training phase is finished, the performance of the Helmholtz machines is tested by presenting an input pattern to each Helmholtz machine and computing the description length needed to represent that input pattern. The class of an input pattern is assigned by the Helmholtz machine that can do this with the least amount of bits, or the shortest description length.

The number of input patterns that are correctly classified by a readout mechanism, is a performance measure that can be used to compare the two mechanisms with each other. However, the performance in the experiments is not defined as the absolute number of correctly classified patterns, but as the percentage of correctly classified patterns with respect to the total amount of patterns in each class. Subsequently, the percentage of correctly classified patterns in each class is averaged over all classes to get a performance measure that can be represented as a single number.

An important aspect of a learning task is the representation of the input patterns to the algorithm, and the preprocessing methods that are involved in creating the representation. In this project the question is: in what form should the musical content of the Well-Tempered Clavier and the Piano Sonatas be provided to the network? To find an answer for this question, the representation of music in the brain has been a major guideline. It was pointed out in the previous chapter that music recognition can be seen as the process of recognizing certain features in music from the low-level preprocessing of sensory input to pitches, to high-level feature extraction such as the recognition of specific melodic and rhythmic structures. When the brain is presented with music, it immediately transforms the different frequencies of which it consists to pitches. There are specific neurons that fire when a certain pitch is detected, and these neurons are ordered by pitch, not frequency, just as the keys on a piano. Modeling such a translation from raw frequencies to ordered pitches would require a sophisticated algorithm. If implemented in an artificial neural network, it would not only require a special structure, training algorithm and a lot of tuning, but also a large amount of computation time. Because the time and resources are limited for this project, we choose not to implement the transformation from frequencies to ordered pitches in our model, but to use music files that contain direct pitch information rather than sound waves.

A commonly used file format for musical files that contains information on the pitch of a note, its onset, its duration and its loudness, is the MIDI format. MIDI files of the Well-Tempered Clavier and the Piano Sonatas downloaded from different sources on the internet are used in this project. Because neural networks cannot process MIDI files, the files are further processed to time series data. For the purpose of the experiments each prelude and fugue from the Well-Tempered Clavier is considered as one musical piece, and for the Piano Sonatas, not the sonata as a whole, but a single movement is considered as one piece.

The data set is divided into two subsets; one part is used during training, and the other part for testing the performance of the networks after training is finished. A neural network learns faster if the data are evenly divided over all classes, so a training set is needed that contains the same number of input patterns in both classes. Book I from the Well-Tempered Clavier and a total of 20 movements from Piano Sonatas 1-15 are used as training set. Piano Sonatas 1-15 last about 4 times as long as book I from the Well-Tempered Clavier, so just one fourth of the movements of the Piano Sonatas are used, to create a training set in which both classes contain an equal number of input patterns. Book II from the Well-Tempered Clavier and 30 movements from Sonatas 16-31 are used as test set.

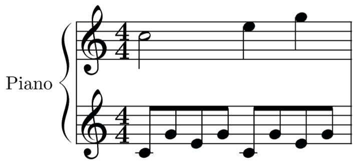
Now that the basic properties of the music classification task are discussed, we define the problem to be learned as "the task of choosing the right class for a MIDI file taken from the Well-Tempered Clavier or Beethoven's Piano Sonatas". For the sake of simplicity, the classes are labeled 'Bach' and 'Beethoven' from now on. To give a general idea about the experiments that are performed, an overview of the different components that are implemented and how these are linked to each other will be given first, while further details of the components are given in the remainder of this chapter.

During the training phase the MIDI files are preprocessed for use with neural networks, and presented to the liquid filter. At certain time steps the liquid states are processed further to a representation that can be handled both by the multilayer perceptron and the Helmholtz machine. Next, this data together with the corresponding target values are used repeatedly to train both readout mechanisms for a certain number of epochs. When the results of the learning process are satisfactory, the training is stopped and the weight and bias values are saved. To test the performance of the networks on novel data, the test set is preprocessed in the same way as the training set and presented to the liquid filter. The liquid states at certain time steps are processed in the same way as the training set, and presented to both readout mechanisms. Next, the results of the readout mechanisms are compared to the actual classes. In this way a comparison between the performance of both readout mechanisms can be made.

6.3 MIDI to time series data

The liquid filter implemented as a spiking neural network, cannot process raw MIDI files. Therefore some basic preprocessing must be performed on the MIDI files, to create a time series data set that can be processed by the liquid. A MIDI file consists of *events*. There are a large number of events that can be represented in such a file, most of which have to do with specific properties of electronic instruments, but are not relevant to this project. The events used here are: the onset and duration of a note, and its pitch. Because the files are played by different artists, or sometimes directly entered into a computer program, it is likely that the contents of the file represents certain musical aspects and preferences of its creator. As the goal of the classification task is to specify the composer, not the musician that played the piece, these effects are removed by neglecting information on dynamics and exact note onsets and durations. Each note is set to the same velocity value, and the beginning and duration of the notes are set to discrete time steps with intervals of $\frac{1}{32}$ note. This value is chosen because any note with a duration shorter than $\frac{1}{32}$ has little implication for the basic melodic line, and is more often used as 'ornament'. Furthermore, all files are transposed to the same key, namely the key of C-major for pieces in a major scale, and A-minor for pieces in a minor scale, such that classes cannot be determined based on key. In a MIDI file a range of 128 different pitches can be represented, but because the data set contained files composed for keyboard instruments, notes that cannot be played or are rarely played on such instruments, are filtered out. After preprocessing the MIDI files, the results are represented in the form of a time series data set, in which the time is divided into discrete time steps of $\frac{1}{32}$ note. Each input pattern in the time series data consists of a set of 72 numbers corresponding to the 72 most-used keys on a piano (starting at f2), where the value of each number is 1 if the corresponding note is 'on' and 0 if the corresponding note is 'off'. A simplified version of the process of converting MIDI files to time series data is depicted in figure 6.1. The total number of input patterns resulting from the process of converting MIDI files to time series data, was about 130 000 in the training set, and about 220 000 in the test set.

In music perception the activity at the onset of a note is of much more importance than the activity during the rest of a note. This can easily be implemented by setting the value of the corresponding note in the time series data to 1 only at the onset of a note and to 0 during the rest of the note, but this results in a data set in which the activity is sparse and has almost no effect on the activity in the liquid. Therefore the values for the corresponding notes are set to 1 for the entire duration of a note in the time series data set. In this fashion the melodic and rhythmic features are explicitly present. In contrast to the brain where certain harmonic



$$\begin{array}{ccccccc|c|}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{array}$$

$t \longrightarrow$

Figure 6.1: Representation of MIDI file in staff (top), and part of the time series data (bottom). The numbers within the dashed box represent an input pattern to the liquid filter at one time step.

relations between different pitches are extracted represented in specific areas, we choose not to extract harmonic features and represent them explicitly, because we want the learning task to be focussed on extracting temporal dependencies. The other choices that are made in the process of converting MIDI files to time series data that can be processed by a spiking neural network as described before, are listed in table 6.1 as parameters with their corresponding values.

Parameter description	**Symbol**	**Value**
lowest note		f2
number of notes		72
resolution		$\frac{1}{32}$ note

Table 6.1: Adjustable parameters for MIDI to time series conversion

6.4 Liquid filter and liquid states

The type of liquid used in the experiments is a spiking neural network as described in section 3.3. The structure of the liquid is a three-dimensional space, in which each unit is located at the integer points. The time series data extracted from the MIDI files consisting of information on the presence of each of the 72 notes are presented to the leftmost column of the network each time step. The dimensions of this column are 12×6 to represent the 12-note scale property of the music used in this project over 6 octaves (= 72 notes). Each unit in a horizontal row of 12 units corresponds to a note in the 12-note scale, and each vertical row to one octave. For each note that is played in the MIDI file at a certain time step, the output of the corresponding unit is set to 1, while the output of the other units is set to 0. To create a three-dimensional space, 5 additional columns of the same size are placed next to the input column. The resulting three-dimensional space of units has dimensions $12 \times 6 \times 6$.

The units in this space are connected to each other according to a stochastic process such that units that are close to each other have a higher chance to be connected than units that lie farther from each other. To create such a network with mainly locally connected units, the chance $p_c(i,j)$ for each unit i that it has a connection to unit j is computed according to:

$$p_c(i,j) = \begin{cases} Ce^{(D(i,j)/\lambda)^2} & \text{if } i \neq j \\ 0 & \text{otherwise} \end{cases}, \tag{6.1}$$

where $D(i,j)$ is the Euclidean distance between units i and j. λ is the parameter that controls both the probability of a connection and the rate at which the probability that two units are connected decreases with increasing distance between the two. C is the parameter that controls just the probability that two units are connected, and can be used in combination with λ to scale the proportion between the rate at which the probability that unit i and j are connected decreases with increasing distance between i and j, and the actual probability of a connection. The second term of equation 6.1 is added to prevent units from having self-recurrent connections. Maass et al. (2002) found that the connectivity in biological settings can be approximated by setting the value λ to 2 and C to a value between 0.1 and 0.4, depending on whether the concerning units are inhibitory or excitatory. We use the same value of λ in our experiments. The value of C is 0.4 for all units, because a distinction between inhibitory and excitatory is not made at the level of units, but at the individual connections.

A connection between unit i and unit j has a certain weight, with value w_{ij}. A certain proportion p_i, of the weights is set to a negative value to provide for inhibitory connections, while the remaining weights are set to positive values, representing excitatory connections. The dynamics of the spiking units were described by equations 3.1, 3.4 and 3.5 in section 3.3. The adjustable parameters in these equations are the half-time of the membrane potential r^h, the absolute and relative refractory period r_i^a and r_i^r of each unit i, and the firing threshold η. The firing threshold is set to 1 for all experiments, and the half-time of the membrane potential to 4 time steps. The absolute refractory period r_i^a, is for each unit i sampled from a Gaussian distribution with mean r^a and standard deviation of $\frac{1}{2}r^a$. The relative refractory period r_i^r, is not implemented as a number of time steps, but as a negative value of the membrane potential after the absolute refractory period is over. In this way it is harder for a unit to reach the threshold value for a certain number of time steps. The relative refractory period depends on the value of the absolute refractory period, and the membrane potential decay constant τ, in the following way:

$$r_i^r = -\frac{\tau r_i^a}{r^s}, \tag{6.2}$$

where r^s is a scale factor which has a value of 10.

When time series data of input patterns are provided to the network, the state $y_i(t)$, of unit i in the network is computed at each discrete time step t, using the states of connected units at the previous time step. The incoming signals propagate through the network, and produce certain activities in the liquid units. The total activity of all units in the network at a certain time step contains information on the inputs that were provided during the previous time steps. It is not necessary to present the liquid state to the readout mechanism at each time step, because the liquid can preserve information on previously seen inputs over several time steps. Some preliminary analysis of the liquid and learning task used in this project, showed that the liquid in most cases returns to a level of stable activity in not more than 100 time steps after the last input is presented. This means that this type of liquid 'forgets' about inputs that have been presented more than about 100 time steps ago when no additional inputs are presented. Of course this isn't a precise measure of the memory capacity of the liquid, but rather an indication of the number of time steps that can reasonably be used as interval between successive readouts. The interval between successive readouts t_v, is set to 128 for all experiments in this project. To give an idea about the duration of the music that is presented to the liquid during this period; 128 time steps correspond to $\frac{128}{32}$ notes in the MIDI file (= 4 bars in $\frac{4}{4}$ measure). The total number of input patterns to the neural networks extracted from the liquid states, was 536 in class 'Bach' and 539 in class 'Beethoven' for the training set, and 816 in class 'Bach' and 980 in class 'Beethoven' for the test set.

At the beginning of the processing of each file, the activity in the network is reset to zero. Because it takes some time before the activity has spread from the input units over the rest of

Parameter description	**Symbol**	**Value**
readout interval	t_v	128
warmup period	t_w	64
number of time steps over which the activity of a unit is summed up during readout	t_a	test
dimensions		$12 \times 6 \times 6$
connectivity parameter	λ	2
connectivity parameter	C	0.4
proportion of inhibitory connections	p_i	test
firing threshold	η	1
mean of the Gaussian distribution from which the absolute refractory period is sampled	r^a	test
scale factor for computing the relative refractory period	r^s	10
half-time of membrane potential	r^h	4
weight between unit i and unit j	w_{ij}	test

Table 6.2: Adjustable parameters for the liquid and liquid states

the network, the collection of liquid states is started after a certain number of time steps. The period between reset at the beginning of the processing of each file and the first readout time is called the *warmup period*, and will be denoted with the symbol t_w. The warmup period is chosen to be half as long as the readout interval.

At the readout moments the activity in the network representing information about present as well as past inputs can be used for further processing by the readout mechanisms. However, because the liquid units are not firing most of the time, the activity in the network is sparse. That means that little information can be extracted from the activities at just one time step. Therefore not the information whether a unit is firing or not at a single time step, but the number of times a unit fires during a certain readout period is used as measure of activity in the network. Because the Helmholtz machine can only process inputs that are represented as patterns of ones and zeros, the total activity of a unit during the readout period is compared with a certain threshold. If the number of times a unit fires during the readout period is above this threshold, the input to the corresponding unit of the readout mechanism is 1, and if it is below this threshold, it is 0. If the readout period is chosen sufficiently small, there will still be a reasonable amount of units that show no activity at all during this period, which means that the threshold value can be set to 0, so no additional adjustable parameter for the threshold value is needed:

$$x_i(t) = \begin{cases} 1 & \text{if } \sum_{t'=t-t_a}^{t} y_i(t') > 0 \\ 0 & \text{otherwise} \end{cases}, \tag{6.3}$$

where t is the time step at which the liquid state is read, $x_i(t)$ is a measure of activity of unit i, and t_a it the number of time steps in the readout period. All adjustable parameters discussed in this section are given in table 6.2. In case the value of a parameter is not given in this section, experiment have to be performed to find an optimal value, which is denoted with the word ‘test’ in the table.

6.5 Readout mechanisms

The liquid state produced by the liquid at a certain time step can be used by a readout mechanism for classifying the successive input patterns that were the cause of that state. The readout mechanisms that are used in the experiments are the Helmholtz machine, and a multilayer perceptron. The architecture of the Helmholtz machine and the wake-sleep training algorithm were given in chapter 4 and the description of the multilayer perceptron and the backpropagation training algorithm in section 2.3.

The wake-sleep algorithm is an unsupervised learning method. However, the minimum description length principle can be used as a performance measure, using a Helmholtz machine for each class of the classification task. To be more specific: there is one Helmholtz machine that has the task to find the best representation for liquid outputs caused by input patterns from Bach, and another machine for Beethoven. The performance of this method can be measured by providing a liquid state to each Helmholtz machine. The machine that can represent that liquid state using the minimal description length corresponds to the class that is assigned. Because the activation function of a Helmholtz machine is of a stochastic nature, the input patterns are provided a number of times and the average description length is taken as measurement and used for comparison. In the experiments the number of times each input pattern is provided to a Helmholtz machine is 10.

The multilayer perceptron can be trained by the supervised backpropagation training algorithm. This algorithm uses an error at the output units and propagates this information back through the network. To compute an error at the output units, a representation of the desired output has to be created. Therefore a multilayer perceptron with two output units is used in the experiments, in which each output unit corresponds to a class. The desired activity of the output unit for the correct class is 1, and 0 for the other classes. When a liquid state is provided to the network and the activities of the units are computed, the output unit with the highest activity corresponds to the class that is assigned by the network.

Because the multilayer perceptron and the Helmholtz machine are based on different principles, it is not immediately obvious what the structure of the networks should be in terms of the number of hidden units or the number of adjustable weights and bias values, for a fair comparison between the two. A Helmholtz machine with the same number of units as a multilayer perceptron uses twice the amount of weights and biases as the multilayer perceptron, because it has separate generative and recognition models. Furthermore, a single multilayer perceptron with a supervised training algorithm is used for all classes, while a Helmholtz machine with unsupervised learning is used for each class. However, the property that both neural networks have in common is that they build a *representation* in the activity of the hidden units of the input patterns that are provided to the input units. Because the ability of a network to build a representation of the liquid states in terms of the activity of the output units or the description length is tested, the number of hidden units in which this activity is represented is the same for all networks.

The results of equation 6.3 for all liquid units are presented to the input layer of the readout neural networks. All units other than the input units of the liquid are used for readout. The dimensions of the liquid without the input units are: $12 \times 6 \times 5$, so the number of input units of the readout neural networks is 360. The multilayer perceptron has a hidden layer, and an output layer with 2 units, while the Helmholtz machine has no output layer. However, as discussed before, the number of units in which a representation of the input can be made is kept the same for all networks, so the Helmholtz machine has two hidden layers where the top hidden layer also has 2 units. The optimal number of hidden units in the first hidden layer for both networks is tested in the experiments.

The training and test sets for the readout mechanism consists of patterns of ones and zeros computed according to equation 6.3. Readouts are performed each 128 time steps for all units in the liquid other than the input units. Because training a neural network while providing the training set just one time often doesn't result in good performance, both networks are trained on the training set several times. One epoch consists of the presentation of all input data to the network while training the network on this data. Neural networks often have to be trained for several epochs before reasonably good results are achieved. However, when a neural network is trained on the training set for too long, it is possible that the network *overfits* the training set. When a neural network is trained it learns to extract the general features of a class, but when training is repeated too often, the network learns to extract features that are specific for the training set but not for the class as a whole. It is often the case that after a number of epochs the performance of the network on the training set still increases, while the performance on the test set starts to decrease due to overfitting. The moment at which the performance of the training algorithm on the test set starts to decrease, is taken as the stopping criterion for the training process. The number of epochs that is required to train the network such

Parameter description	**Symbol**	**Value**
number times an input pattern is provided to a Helmholtz machine to compute the average description length		10
number of units used for readout (= number of units in the input layer)		360
number of units in first hidden layer		test
number of units in second hidden layer / output units		2
interval of the uniform distribution from which the weights of the multilayer perceptron are sampled at initialization		$[-0.3, 0.3]$
interval of the uniform distribution from which the biases of the multilayer perceptron are sampled at initialization		$[-0.1, 0.1]$
initial values of all weights and biases of the Helmholtz machine		0
learning rate for Helmholtz machine	γ	0.005
learning rate for multilayer perceptron	ϵ	0.0001
batch size		10
number of training epochs for the Helmholtz machine		test
number of training epochs for the multilayer perceptron		test

Table 6.3: Adjustable parameters for the readout mechanisms

that the performance on novel data is optimized is not a priori known, and has to be found experimentally.

During training the performance of the neural networks is improved by adjusting the weights of the connections between units and the bias values of the units. In equations 2.11, 4.19 and 4.20 the rules for the weight updates for the multilayer perceptron and the Helmholtz machine were given. These equations include the learning rate parameters ϵ and γ for the multilayer perceptron and the Helmholtz machine respectively. As pointed out in section 2.3, the learning rate parameter has to be chosen carefully. In a preliminary analysis several learning rate parameters were tested for both types of networks. When the value of the parameter was too large, the error or description length did not decrease steadily, but oscillated between different values, and when the learning rate was too small, it took a very large number of epochs before reasonable results were attained. The training algorithms were however not very sensitive to the exact value of the learning rate parameters, such that a difference of up to a factor ten didn't have much impact on the final performance. Unless otherwise specified, the values of the learning rate parameters ϵ and γ were set to 0.0001 and 0.005 respectively.

When a neural network is initialized, the weights and bias values have to be set to certain values. The weight and bias values for the Helmholtz machine were all set to 0, as is often done with this type of neural network (Dayan et al., 1995). The weights and biases of the multilayer perceptron were sampled from a uniform distribution with intervals $[-0.1, 0.1]$ for the biases and $[-0.3, 0.3]$ for the weights. Next, the training algorithm computes the weight updates that are required to better match the target or decrease the description length, given an input pattern. It is possible to update the weights each time an input pattern is provided to the network, but the weight updates can also be summed over the presentation of a number of input patterns. The weight updates are then applied after the presentation of a certain number of input patterns. In this fashion the update process is 'smooth', and the chance that weights start to oscillate between different values is diminished. The number of input patterns that are provided to the network until the weights are updated, is called the *batch size*. In all experiments the number of input patterns per batch is set to 10. Furthermore, the order in which the patterns were provided to the readout mechanisms is randomized at the beginning of each new epoch, which improves the learning speed and makes it less likely that the training algorithm gets stuck in local optima. The adjustable parameters for the readout mechanisms as discussed in this section are given in table 6.3. Again the value 'test' means that the optimal value has to be found empirically.

Chapter 7

Experiments and results

7.1 Overview

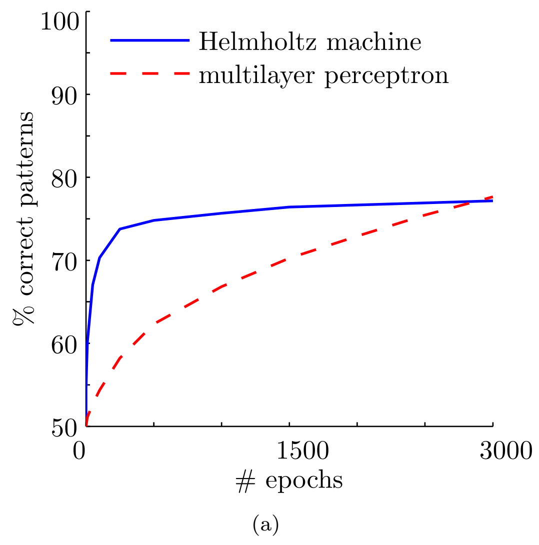
Since there are a number of parameters of the liquid state machine, the Helmholtz machine and the multilayer perceptron that cannot just be copied from existing literature or set by educated guesses, experiments were performed to examine the performance on the learning task for different parameters settings. The neural networks that were used as readout mechanisms have to be initialized at the beginning of the learning process, which involves setting certain values according to a stochastic process. For the multilayer perceptron the initial weights and biases have to be set to small values from a uniform distribution, and the Helmholtz machine uses a stochastic unit activation function all the time. Because of the stochastic nature of the initialization and computation, training networks that are initialized with different random values can produce different results, so the results of a single trained network cannot be used as a reliable method. Therefore in all experiments the performance was measured for ten networks with different initialized values, and the *average* performance over these ten networks is shown in the results.

The results of most experiments are shown graphically, which does not only give a general overview of the effect that changing the parameter has, but also allows us to see at which value of the parameter the best performance is achieved. Each experiment yields an optimal value or range of values for the parameter that is tested, which is used in later experiments. In this fashion, the performance on the learning task will increase in each consecutive experiment. It is however not guaranteed that an optimal performance is achieved for this setup, because each parameter was tested only in one experiment and the effect that changing another parameter in further experiments had on the optimal value of the parameter in previous experiments was not taken into account. Testing each parameter over and over again when another parameter is changed will require an enormous amount of experiments. Because the time for this project is limited, we have thought carefully about the order in which the experiments were performed, and which parameters are the most important, such that the final performance of the setup is hopefully close to optimal. Besides, the precise value of a parameter is not of great importance; changing a parameter by a relatively large factor has often little effect on the final performance.

Because the performance of the readout mechanisms was used as the criterion for finding the right parameter settings for the liquid, the optimal parameter settings of the readout mechanisms were investigated first. The optimal parameters for the liquid filter were not yet known at this stage, so they had to be found by running some preliminary experiments to determine at which value of the parameters the liquid produced 'sensible' liquid states. Because not much can be learned if the input patterns to the readout mechanisms are all the same, the separation property of the liquid filter for various parameter settings was tested by investigating the number of liquid states that were the same. By using this very simple method, we found that the following parameters for the liquid produced reasonably different liquid states: $(p_i = 0.2, r^a = 4, w_{ij} = 0.4, t_a = 12)$. Furthermore, all units of the liquid filter other than the input units were used for readout, and presented to the corresponding units of the readout mechanisms. The liquid parameters were fine-tuned after the optimal values for the readout mechanisms were found. We expect however that the optimal values of some parameters of the liquid might not be far from the values we found during the preliminary experiments. Since we had to set certain values to the liquid parameters it might not be surprising that these 'guessed' values are also found to be optimal in other experiments because most parameters are interdependent.

7.2 Readout mechanisms

The first parameter of the readout mechanisms that was tested, is the number of epochs a network should be trained. This is very important for the rest of the experiments, because it determines at which moment the training algorithm starts to overfit, and should be stopped. When the performance on the test set starts to decrease after a certain number of epochs, it is unlikely that it will start to improve again, because the algorithm is probably starting to overfit the training set. The moment the process of overfitting begins is used as the stopping criterion for the training algorithm. To get an idea at which point the training should be stopped, several experiments were performed for different numbers of epochs, the result of which are shown in figure 7.1. The other parameters for the networks were as described before, and the number of units in the network was 360 in the input layer, 8 in the first hidden layer and 2 in the output layer or second hidden layer in case of the Helmholtz machine. As can be derived from figure 7.1(a) the performance of both networks on the training set is improving all the time, while the performance of the trained networks on the test set as shown in figure 7.1(b) first increases, but after a certain number of epochs starts to decrease. This is exactly what was expected, and shows that the best performance on the test set can be achieved by stopping at the moment the algorithm starts to overfit.



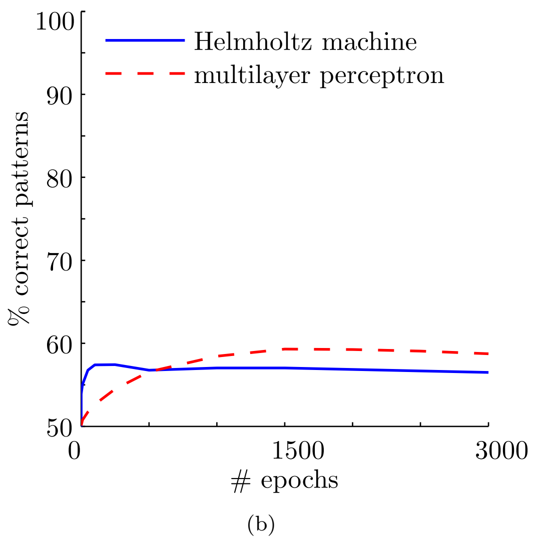


Figure 7.1: Results for different number of epochs on the training set (a), and on the test set (b)

Another aspect that can be deduced from figure 7.1, is that the Helmholtz machine learns much faster than the multilayer perceptron. In other words: the best performance of the Helmholtz machine on the training set is reached in several tens of epochs, while the maximum performance of the multilayer perceptron is achieved in several thousands of epochs. A possible explanation for this behavior is that the learning rate parameter γ of the Helmholtz machine is fifty times as large as the learning rate parameter ϵ of the multilayer perceptron. As described in the previous chapter, the reason that γ is much larger than ϵ is that the largest possible learning rate parameter for a network of a certain size was chosen such that the performance on the training set steadily increases and does not oscillate between different values. To investigate if the difference in learning rate parameters is indeed an explanation for the difference in learning speed, another experiment was performed in which the learning rate parameters of both learning algorithms were set to the same values. Because the learning rate parameter for each network were set as large as possible to obtain a steadily increase in performance, the learning rate parameter of the Helmholtz machine was set to a smaller value, while the learning rate parameter of the multilayer perceptron was not changed, so: $\gamma = \epsilon = 0.0001$. The other parameters were the same as in the previous experiment. From the results in figure 7.2 can be concluded that the Helmholtz machine still learns faster. It is however interesting to notice that the maximum performance on the test set achieved by the Helmholtz machine is not as good as in the previous experiment. Although the value of the learning rate parameter is an important factor, the difference between the learning speeds between the Helmholtz machine and the multilayer perceptron can not only be attributed to the difference in learning rate parameters. The number of epochs at which the best performance on the test set was achieved, is used as stopping criterion for the experiments from now on, which is 200 for the Helmholtz machine and 1750 for the multilayer perceptron.

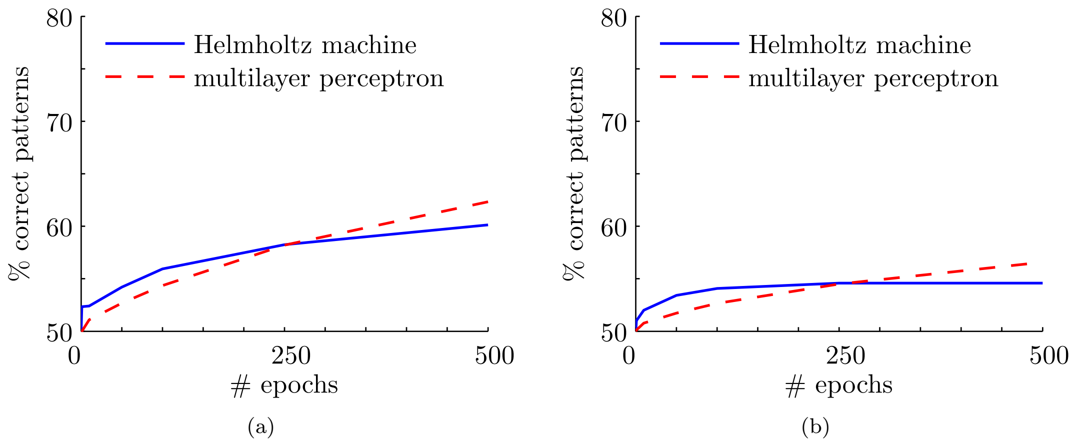


Figure 7.2: Results for different number of epochs on the training set (a), and on the test set (b) where both networks were trained with the same learning rate parameters

Now that the stopping criterion has been established, another important question is how large the readout neural networks should be in terms of number of hidden layers and the number of units in these layers. The more hidden units the network has, the better it can learn to extract features from the input patterns. On the other hand, the risk of overfitting increases with more hidden units. When a network has more hidden units, the learning rate should be set to a smaller value which results in a larger number of epochs that is needed to train the network. It is often the case that a certain learning task requires a minimum amount of hidden units, while increasing the number of units too far comes at a great cost in terms of overfitting and learning speed. To get an idea about the number of hidden units that is sufficient for this task, a number of experiments were performed with different numbers of hidden units in the readout networks. An additional experiment was performed in which the networks had no first hidden layer at all. The results of the experiments are depicted in figure 7.3. From this figure can be inferred that the performance on the test set increases first with an increasing amount of units in the hidden layer, but slowly starts to decrease when the number of hidden units gets larger than about 10. The performance on the training set (which is not shown here) increases all the time, so the decrease in performance on the test set when the number of hidden units is more than 10, is not because the network needs to be trained for more epochs, but is probably due to overfitting. The best performance was achieved in a network with a hidden layer with 10 hidden units, so the size of the networks that will be used from now on is 360 units in the input layer, 10 units in the first hidden layer and 2 units in the output layer or second hidden layer in case of the Helmholtz machine.

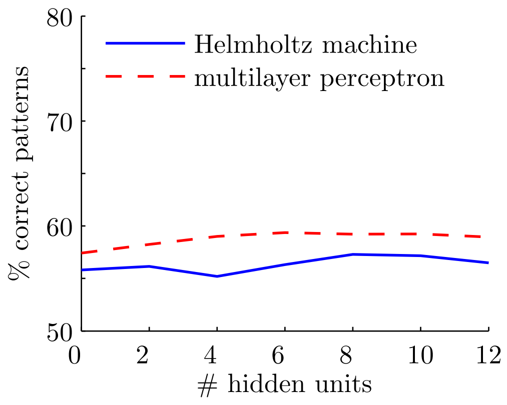


Figure 7.3: Performance on the test set for different number of hidden units in the first hidden layer. 0 units means no first hidden layer at all.

7.3 Liquid optimization

The results of the previous experiments are well above guess-chance, but as the parameter settings of the liquid are initial guesses, the performance can hopefully be improved by analyzing the dynamics of the liquid more closely. The performance of the readout mechanisms were averages over ten randomly initialized networks, while just one liquid was used for generating the liquid states. It might be the case that a differently initialized liquid yields other results. To examine the variation between the results of different randomly initialized liquids, the average performance of ten networks for ten different liquids was investigated. The results of this experiment as depicted in figure 7.4 show that the difference between the best and worst performance for both readout mechanisms is not more than 2%. Testing all liquid parameters for a number of different liquids, and training 10 different instances of each readout mechanism would take a very long time. Because time and computational resources are limited for this project and the difference in performance between different instances of the liquid is not much, all liquid parameters will be tested just for one instance of the spiking neural network.

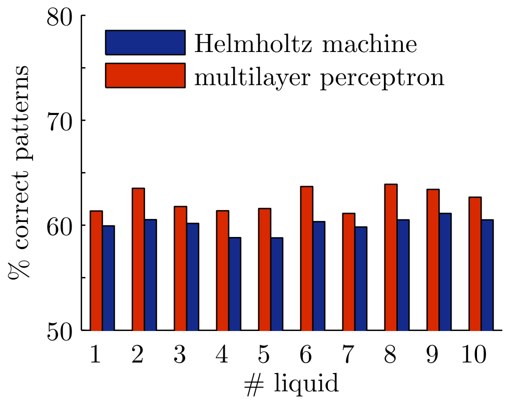


Figure 7.4: Performance of readout mechanisms on the test set for liquids with different random seeds at initialization

Two important parameters that determine the activity of the units in the liquid are the proportion of inhibitory connections p_i, and the value of the weights w_{ij}. First the best value for p_i was determined by measuring the performance on several different values, while the other parameters were: $(r^a = 4, w_{ij} = 0.4, t_a = 12)$. The result of this experiment are shown in figure 7.5(a). The value of p_i in the experiments performed till now was 20%, but from figure 7.5(a) can be deduced that increasing this value to 30% largely improves performance, so p_i will be set to 30% in all experiments from now on.

Next, the performance for different values of the weights of the connections between the liquid units was tested, the result of which is depicted in figure 7.5(b). The other parameters were: ($r^a = 4, t_a = 12$). The weight value for which the best performance is achieved, is 0.4, which is not surprising because this value was also used in the previous experiment to determine the optimal value for the percentage of inhibitory connections. The results in figure 7.5 show that the proportion of inhibitory connections and the values of the weights are important factors for the amount of information that can be extracted from the liquid and used for classification by the readout mechanisms.

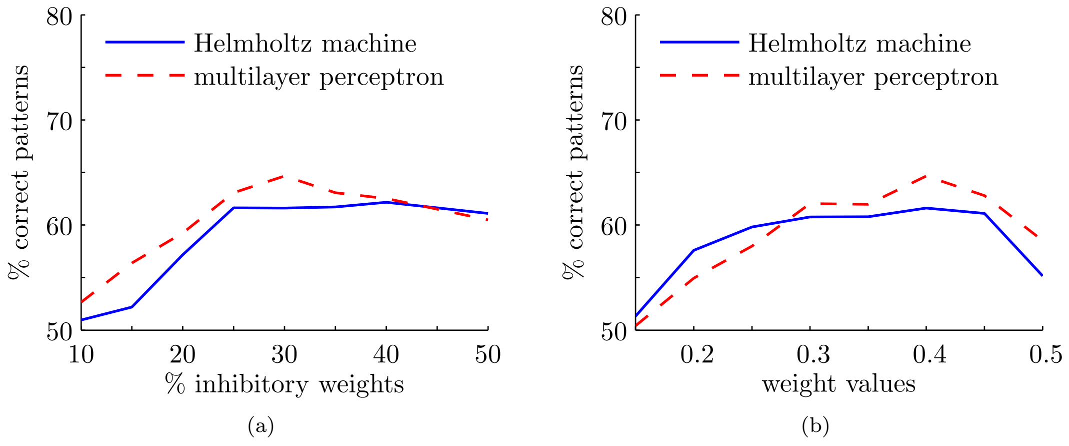


Figure 7.5: Results on the test set for different values for percentage of inhibitory weights (a), the values of the weights (b)

The next parameter that was tested, was the value of the mean of the Gaussian distribution from which the absolute refractory period is sampled; r^a. The value of the relative refractory period r^r also depends on r^a. As given in equation 6.2 a larger larger value for r^a results in an increase in relative refractory period, so changing r^a affects both the relative and absolute refractory period. The optimal values of the other parameters were specified before, except for t_a, which was set to 12. The results depicted in figure 7.6 show that the actual value of r^a is not of much importance, as long as it is between about 2 and 7. The value that was used for r^a was 4 in other experiments. This value also yields good results in this experiment, so we will continue to use this value.

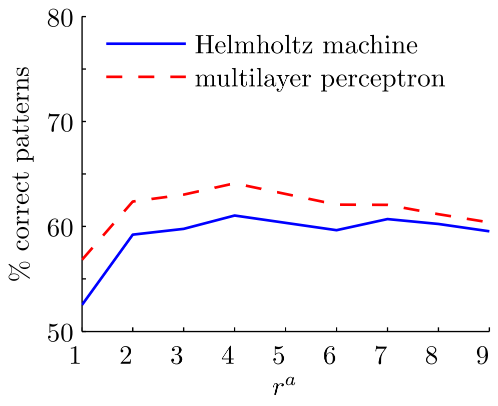


Figure 7.6: Results on the test set for different values of the absolute refractory period

The last parameter that was tested is the number of time steps over which the activity of a liquid unit is summed during readout. Figure 7.7 shows the performance on the learning task for the test set for different values of this parameter. As it turns out, the number of time steps over which the activity of a liquid unit is summed during readout is of great importance for the performance on the task. The performance continues to increase until the activity over the entire readout interval (128) is summed up. In equation 6.3 the summed activity of the liquid units over the readout period is compared with a threshold value which was set to 0 to avoid the introduction of an additional parameter. In this fashion the input to the corresponding unit in the readout mechanisms is 1 when a unit in the liquid fires during the period that the liquid activity is read, and 0 if the liquid unit does not fire. It might be that the value of this threshold parameter also affects the performance. Several additional experiments were performed to investigate the importance of the threshold value in equation 6.3. These experiments of which the results are not graphically shown here did not yield better results for other values of the threshold parameter. That means that it is much more important for the readout mechanisms to know that the corresponding unit in the spiking neural network didn't fire *at all* than the difference between the number of spikes that were produced by units that fired one or more times.

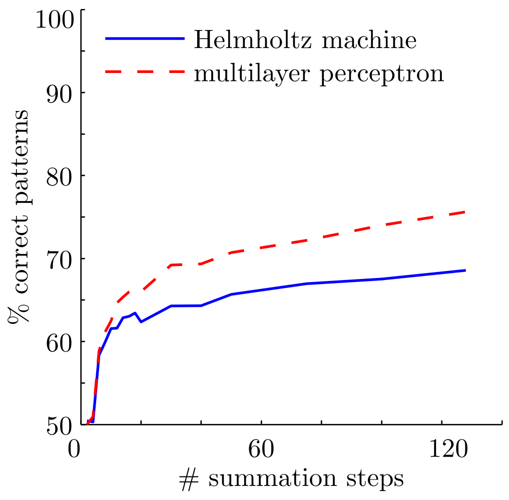


Figure 7.7: Results for different values of summation steps for the test set

The results in figure 7.7 also show that the performance on the task largely depends on the summation of the liquid states. This summation is just a method used to obtain a measure for the activity in the liquid, while it is not involved in the dynamics of the liquid itself. Because it seems that the number of steps over which the summation is performed is of great importance for the performance on this task, it is interesting to examine if the learning task can also be solved with a summation function only, without using the spiking neural network. If it is the case that the summation over the liquid states is much more important than the dynamics of the liquid, a summation of the inputs over the same number of time steps would produce comparable or even better results. To test this hypothesis, an experiment was performed in which the summation was computed over the input patterns instead of the liquid states. The summation over the inputs was preprocessed by the threshold function from equation 6.3 before presenting it to the readout mechanisms, in the same fashion as the summation over the activities of the units in the liquid filter. The results of this experiment together with the results of the task *with* a spiking neural network as liquid are shown in figure 7.8. It can be concluded from this figure that it is indeed the case that the summation is the most important factor while the use of a liquid only reduces performance.

Now that we have shown that a very simple temporal filter that computes the sum of the input patterns over several time steps performs better than a more complex filter such as a spiking neural network, the question is why the spiking neural network isn't of much use. The previous experiment showed that the number of time steps over which the activity of the liquid units is summed up is more important than the liquid dynamics. That means that the summed activity of the unit does not contain much information on the activity of the liquid outside the summation period. If for example the number of steps over which the activity of the liquid is summed up is 12, the information of the last 12 time steps is available, but nothing can be learned from the summation over the last 12 time steps about the activity in the liquid

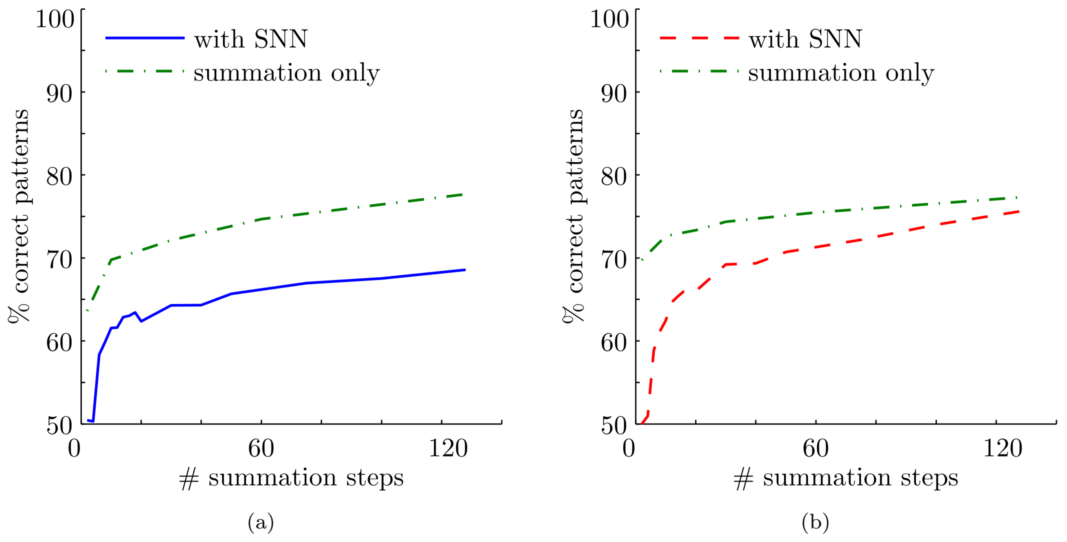


Figure 7.8: Results for different values of the number of inputs or liquid states over which the summation function is computed, for the Helmholtz machine (a), and the multilayer perceptron (b)

before those 12 steps. Even with the optimal parameter settings that were found empirically, the liquid cannot retain much information over time. A possible reason for this is that the input does not contain enough activity to cause a significant disturbance in the liquid state. The input patterns to the spiking neural network are the result of the translation process of the MIDI files to time series data. In the MIDI file the number of notes that are played at a certain time, is often around four, and not more than ten. The same number of liquid units is activated in a single time step by the input patterns. A possible solution to the small number of perturbations of the liquid, is to copy the activity of each note to the corresponding notes in other octaves. For example, if the note 'g4' is played in the MIDI file, the input units of the liquid that correspond to 'g2', 'g3', 'g5', 'g6' and 'g7' are also activated. The drawback of this method is that only twelve notes can be differentiated, which means that the input patterns contain less information. This representation can however be used to demonstrate whether the

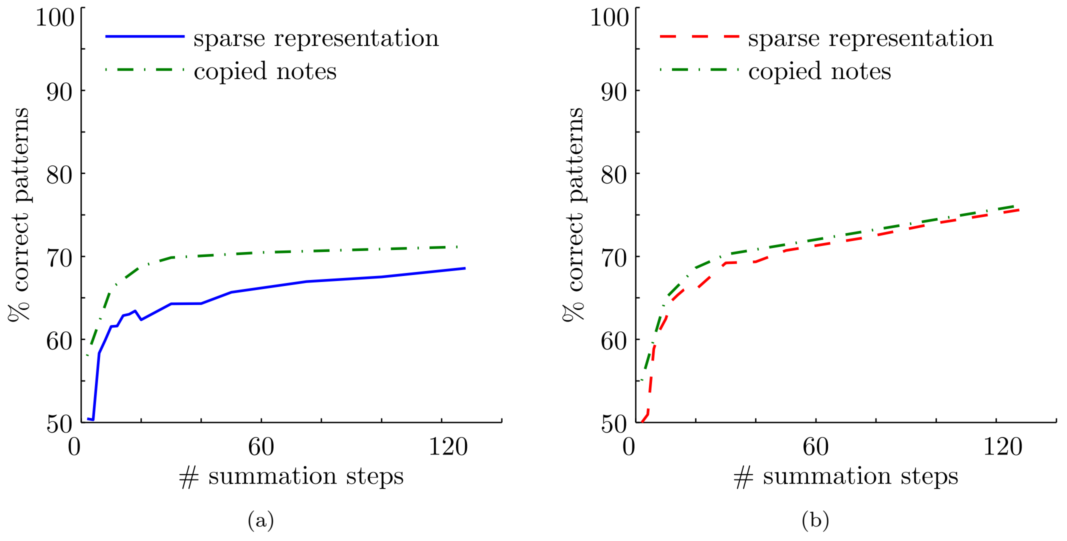


Figure 7.9: Results for different representations of the input patterns over several summation steps for the Helmholtz machine (a), and the multilayer perceptron (b)

fact that the liquid cannot retain much information over time is due to the sparseness of the activity in the input patterns.

Figure 7.9 shows that the representation in which the notes are copied to the same notes in other octaves yield better results than than the sparse representation that was used in previous experiments. That means that although the representation with copied notes was less information-rich, the increased activity in the liquid enabled better performance on the task. Therefore it is likely that the difference in performance between the liquid and a simple summation function is at least partially caused by the sparse activity in the representation of the learning task.

7.4 Improving performance

In section 7.2 the point at which the networks began to overfit was empirically determined to be 200 and 1750 epochs for the Helmholtz machine and the multilayer perceptron respectively. Because a lot of parameters have been changed, it might be the case that the moment that the networks start to overfit has changed. Therefore the experiment in which the optimal number of epochs is determined is repeated with the parameter settings of the spiking neural networks that produced the best results. The results in figure 7.10 show that the optimal number of epochs has indeed changed. The best results are obtained at 40 epochs for the Helmholtz machine and 2000 for the multilayer perceptron.

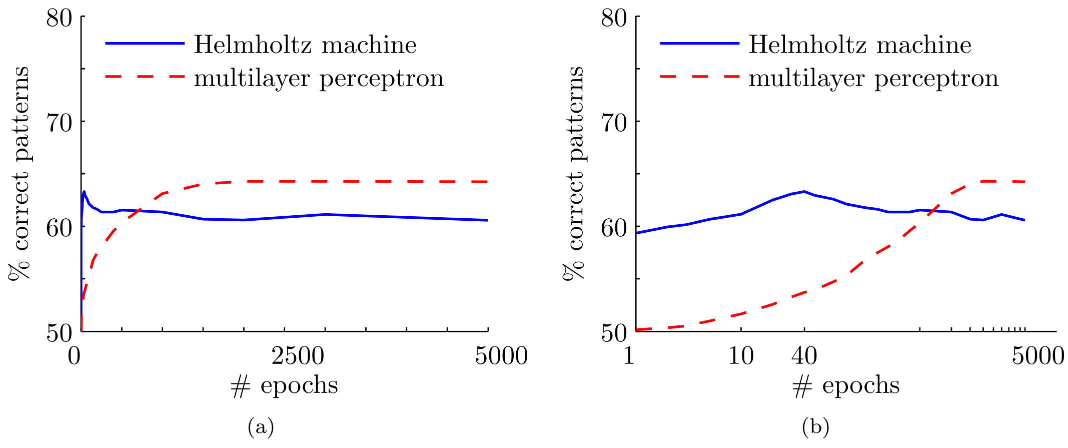


Figure 7.10: Results for different number of epochs on the training set retested on optimal liquid parameter settings (a), and the same results with a logarithmic scale on the horizontal axis (b)

In the experiments described in the previous sections the results of the multilayer perceptron were almost always better than the results of the Helmholtz machine. Although the difference in average performance was small, the Helmholtz machine showed much more variation between the performance over the ten trained instances. Sometimes the results of the best out of ten trained Helmholtz machines were even better than the best out of ten multilayer perceptrons. On the other hand, the Helmholtz machine with the worst results performed almost always worse than the multilayer perceptron with the lowest performance. Because of the relatively large variation in performance between the different instances of the Helmholtz machine, it might be the case that the performance on the learning task can be improved by using the judgement of all individual networks together instead of the average performance over the networks. In this method, which we will call an *ensemble* method, the class that the majority of the networks assigns to an input pattern is taken as the output. To avoid cases in which an equal number of networks assigned different classes, the decisions of nine (instead of ten) networks were used. The average performance of the individual networks for experiment 7.10 at the optimal number of epochs is compared with the ensemble method. The results are given in figure 7.11. From these results it is clear that the performance of both readout mechanisms can

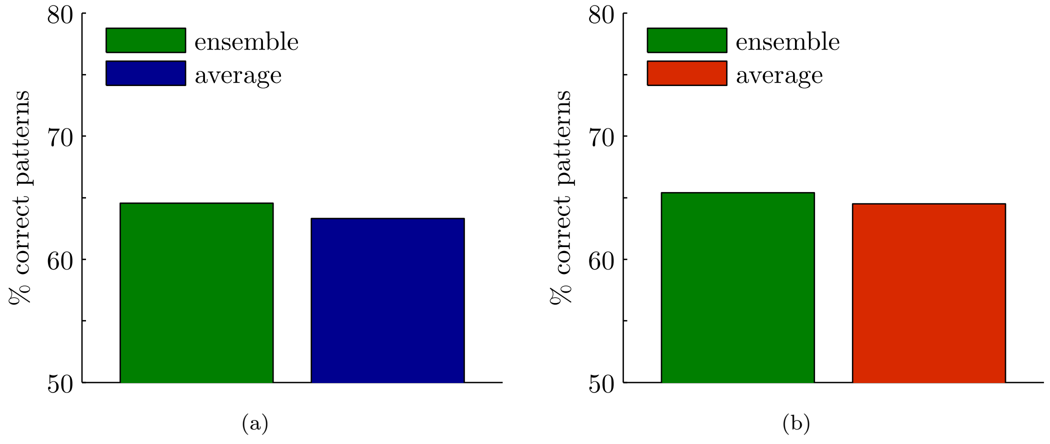


Figure 7.11: Results for ensemble method compared with average performance of the Helmholtz machine (a), and the multilayer perceptron (b)

slightly be improved by using the ensemble method instead of the average results of individual networks. Although the variance in performance between different trained instances of the Helmholtz machine is larger than that of the multilayer perceptron, the variance remains small in both cases.

The performance in the previous experiments has been measured by the percentage of samples that was classified correctly. From this performance measure it is not immediately clear how much of the musical *pieces* were correctly classified. In theory, if the percentage of samples is above 50% and the incorrectly classified samples are evenly distributed over all files, 100% of the files should be classified correctly. To investigate how the correctly classified samples are distributed over the different musical pieces, the number samples that is assigned to each class is counted for a piece, and the class that is assigned to the majority of the samples is taken as the class for that piece. In this method called *majority voting*, it might be the case that the same number of samples is assigned to two different classes. In that case no decision can be made to which class the algorithm assigns that particular piece. Majority voting for the best results of experiment 7.10 is applied, and the results are given in figure 7.12. In this majority voting method 81% of the files are correctly classified by the Helmholtz machine, and 88% by the multilayer perceptron. The percentage of files for which no decision could be made was 5% for the Helmholtz machine and 3% for the multilayer perceptron.

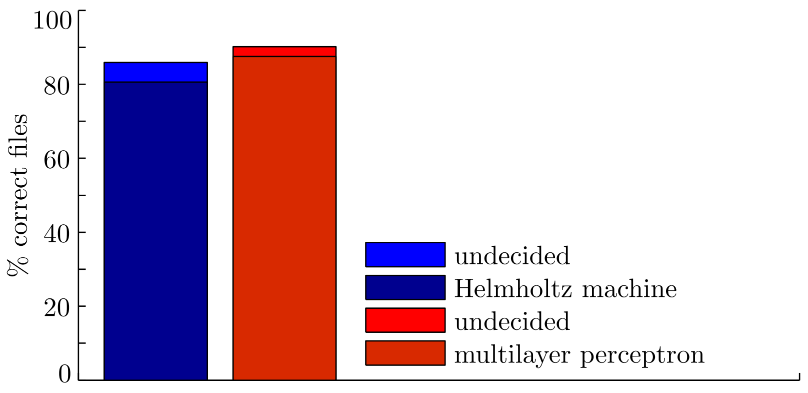


Figure 7.12: Results for majority voting on the test set

Chapter 8

Conclusion

8.1 Discussion

The experiments as described in the previous chapter were performed to investigate the capabilities of the Helmholtz machine as readout mechanism of the liquid state machine, and to compare it with other readout mechanisms. Because the multilayer perceptron can solve a large range of problems, the performance of the Helmholtz machine was compared with that of a multilayer perceptron. The results of the experiments have shed some light on the difference in performance between the Helmholtz machine and the multilayer perceptron, but along the way it became clear that the spiking neural network that was used as liquid did not suffice. A very simple summation function performed better than the more complex spiking neural network. Therefore we will first discuss the different aspects that might be responsible for this behavior.

The learning task could be solved by the neural networks that were used as readout mechanism with a reasonably good performance even without a temporal filter. The average percentage of input patterns that was correctly classified by the Helmholtz machine without a temporal filter was about 64% and 70% for the multilayer perceptron (see figure 7.8 at a single summation step). While this is not perfect, it is possible to improve the performance above these percentages by taking into account the temporal dependencies between successive input patterns. Although the performance is well above guess chance without the extraction of temporal dependencies, this doesn't necessarily imply that the learning task is too simple. Even with the use of a simple summation function as temporal filter the performance did not reach 100%, so it is possible to improve the performance on the learning task by using a more complex temporal filter. This however, was not the case for the liquid used in this project.

There are several reasons for the bad performance of the liquid used in the experiments. The liquid that was used in this project is a simplified version of the spiking neural network that was used by Maass et al. (2002). An important difference is that in contrast to the differential equations used by Maass, our spiking neural network computed the state of the liquid in an iterative manner in discrete time steps. That means that it was not possible to differentiate between delay times of connections. Another difference is that individual connections were chosen to be inhibitory or excitatory, while Maass' spiking neural network consisted of units with only one type of connections. The functions that were used in each unit were the same as in Maass' network, albeit that the membrane potential equivalent and activation values were computed in an iterative manner instead of differential equations. Despite the differences between the spiking neural network used in this project and the one used by Maass, we don't expect this to be the major cause of the bad performance of the liquid. After all, the actual complexity of the liquid is not of great importance, which was already demonstrated by Fernando and Sojakka (2003), who used real water as temporal filter in a speech recognition task.

Another reason which is more likely to be the cause of the bad performance of the spiking neural network is that the activity that was presented to the spiking neural network was too sparse to create a significant perturbation in its stable state. Support for this explanation is provided by the last experiment from section 7.3. In this experiment the notes in the MIDI file were copied to the same location in all other octaves. In this fashion, just the position of a note relative to other notes in an octave was represented, not its absolute pitch. Whereas the information that was available in this representation was less, the fact that more input units of the liquid were activated in a single time step led to better performance. That means the sparseness in the patterns is a probable cause of the bad performance of the liquid. Better results for the music classification task could probably be achieved by using other representations of music files that contain less sparse input patterns, for example by adding explicit information on harmonic relations or by using the Fourier transform of a wave file.

The most important conclusion from the experiments with regard to the liquid filter, is that a complex temporal filter isn't always better than a simple one. Projecting the input patterns into a high-dimensional space should make the problem of finding temporal dependencies easier for a simple readout mechanism. Although the spiking neural network can show more dynamical behavior and complexity than the simple summation function, the summation turned out to be the most important factor for the performance on the learning task. As discussed before, a possible reason for this is that the representation of the learning task was not appropriate for a spiking neural network. Another possibility is that although the liquid state machine is an interesting theoretical invention, it is not clear how it could be applied to problems involving real-world data. Of course it is possible to design a theoretical learning task or toy problem in which the temporal dependencies between successive input patterns can by no means be found by using a summation function, while a spiking neural network still can. We believe however that learning tasks involving real-world data can often be solved by using more simple temporal filters. Others have also used spiking neural networks as liquid in combination with a low-pass filter or other temporal summation functions to convert the liquid states into representations that can be processed by the readout mechanisms (Maass et al., 2002; Vreeken, 2004). It could be the case that the results of their experiments on real-world data should not be ascribed to the complex dynamics of the spiking neural network, but are caused by the temporal summation over the input patterns or liquid states.

Despite the fact that the representation of the learning task in combination with the spiking neural network used as liquid filter did not produce spectacular results, the experiments provide enough information to discuss the difference between the Helmholtz machine and the multilayer perceptron that were used as readout mechanisms. An important question asked in the introduction is whether the Helmholtz machine could extract any information at all from the complex dynamics of a temporal liquid filter. The reason why this question was put forward is that a Helmholtz machine is based on building a generative model of the processes that created an input pattern, whereas the spiking neural network used for preprocessing complicates this process. Although the performance of the Helmholtz machine was mostly worse than that of the multilayer perceptron, the Helmholtz machine was still able to learn this task well above guess chance. Moreover, the graphs in the previous chapter show without exception the performance of the Helmholtz machine depends on the different parameter settings of the liquid in the same way as the performance of the multilayer perceptron.

Another aspect of the Helmholtz machine is the large variation between the performance of different randomly initialized instances with respect to the variation of the multilayer perceptrons. Because the weights and bias values of both the generative and the recognition model were all set to 0 at initialization, the variation in performance between different instances of the Helmholtz machine could only have been caused by the stochastic nature of the unit activation function. Although the results of the best out of ten trained Helmholtz machines were sometimes better than the results of the best out of of ten trained multilayer perceptrons, the average performance over all trained Helmholtz machines was always worse than the average performance of the multilayer perceptrons. Because of the variance between the results of different initialized instances of the Helmholtz machine, an ensemble method was applied to investigate whether the results of the Helmholtz machine could be improved. The results of the ensemble method showed that the performance of both networks could indeed be improved, and that the difference between the performance of the two readout mechanisms is less in the ensemble method. From these results can be derived that the difference in performance as shown in other experiments was probably not caused by the different principles on which the learning algorithms are based, but mainly because of the stochastic nature of the processes that govern the behavior of the Helmholtz machine. The multilayer perceptron learns by extracting features, whereas the Helmholtz machine is based on building models of the input patterns by finding the minimum description length. Despite these different approaches, the experiments have demonstrated that both learning mechanisms can be used to extract information about the temporal dependencies in the input data from the states of a temporal filter.

Although the performance of the Helmholtz machine is slightly less than that of the multilayer perceptron, there is a big difference between the two in terms of learning speed. The best performance of the Helmholtz machine is achieved after only a few epochs, while the best performance of the multilayer perceptron is achieved after several hundreds of epochs. The results in figure 7.2 show that this difference it not only caused by different learning rates, but that the Helmholtz machine in general learns much faster. This is actually a very nice property of the Helmholtz machine, because most artificial neural networks that are used to model certain features of biological neural networks often have to be presented with the training set for several times. In terms of the learning task used in this project that would mean that a person has to learn to discriminate between book II of the Well-Tempered Clavier and Piano Sonatas $15-30$ by listening to book I and Piano Sonatas $1-15$ several thousand times. One could imagine that it is not only highly implausible that anyone would be willing to listen to The Well-Tempered Clavier and the Piano sonatas over and over again, but that the performance of a real person would be better than most artificial neural networks after hearing just a couple of pieces. The Helmholtz machine however, learns most of the characteristics that can be extracted from the training set after just one epoch, while the performance hardly improves any further during additional training epochs, as can be derived from figure 7.10.

Although the performance of the multilayer perceptron is better than that of the Helmholtz machine, it is interesting to notice that the latter learns much faster. Therefore we conclude that the principles of pattern theory which involve not only the extraction of features, but also the reconstruction of the input patterns from the extracted features can successfully be applied to sensory data that are preprocessed by a filter that extracts temporal dependencies between successive input patterns.

8.2 Further research

During the experiments that were performed in this project, some interesting results were obtained with respect to the liquid filter and the Helmholtz machine. In the discussion in the previous section, a number of explanations were put forward which are more or less confirmed by the experiments, but it isn't clear wether these conclusions extend to other learning tasks, and types of neural networks. Some recommendations and possibilities for further research are given here that can be used to investigate the explanations and hypothesis that arose from this project into further detail.

In this project the Well-Tempered Clavier by Bach and the Piano Sonatas by Beethoven were the two classes of which the learning task consisted. The readout mechanisms could solve this task even without a temporal filter, but the performance was not optimal. Although this allows for better results by introducing a temporal filter, the application of a liquid state machine wasn't of much use. A possibility to increase the difficulty of the music classification task, is to use other classes or composers. Pieces of other composers that are more difficult to discern are for instance the String Quartets by Haydn and Mozart. There is a quiz on the internet (http://qq.themefinder.org/) in which the contesters can listen to a random movement in MIDI format, and have to choose the right composer. It would be interesting to compare the results of the setup used in this project with the results achieved by humans on this task.

As described in the previous section, the representation of the MIDI files in combination with the spiking neural network was probably the cause of the bad performance of this complex temporal filter. Because the information on whether a certain pitch is played didn't cause a sufficient amount of disturbance in the liquid states, other representations might perform better in combination with this type of liquid. For example, the density of the representation used in this project could be increased by adding information on the harmonic properties between different notes, which can easily be computed from the MIDI files. Another representation that is often used for classification of audio data, is the Fourier transform of a digital wave file. In such a representation the wave data are analyzed to determine which frequencies or frequency bands are present within a certain time-frame. The Fourier transform of the wave information of a recorded musical piece contains much more information than just the pitches, so it might be interesting to investigate if such a representation would produce better results than the pitch representation used in this project, and whether this is due to the extra information, or is caused by more significant perturbations of the liquid state.

The most important conclusion with respect to the spiking neural network that was used as liquid filter is that complex temporal filters are not necessarily better than simple ones. Moreover, it might be the case that the results of the liquid state machine in this project as well as the results of others for problems concerning real-world data are caused by the low-pass filter that is applied to the activities of the liquid units over time, and not by the complex dynamics of the spiking neural network. Additional support for this idea is found in the experiments of Fernando and Sojakka (2003), in which water was used as liquid filter. As the complexity of the liquid is not of great importance, it might be interesting to investigate whether simple temporal filters such as the summation function used here, or a moving average method, produce comparable results on real-world data. Another method to investigate the use of a spiking neural network, is to compare a representation in which the input to the readout mechanism consists of the results of a summation function, with a representation in which the input consists of the results of the summation function on the input data *and* the liquid states.

The temporal dependencies that can be represented in the state of a liquid highly depend on its structure. Certain parameters of the spiking neural network used as liquid are set according to stochastic processes. The values that result from these processes determine the structure of the liquid, and with that the temporal features that can be represented in the state of the liquid. Differently initialized liquids can represent different temporal features. It might be interesting to investigate the presentation of the input patterns to differently initialized liquids, and train a readout mechanism for each liquid. Instead of using an ensemble method in which the decision of differently initialized readout mechanisms is used, a method in which the decision of the readout mechanisms of differently initialized liquids is used, would probably improve the result.

Whereas learning tasks involving real-world data might be easily solved by simple temporal filters, toy problems such as learning to discern spike trains from different Gaussian distributions as was done by Maass et al. (2002), can shed more light on the actual capabilities of the spiking neural network as liquid filter. Furthermore, from the perspective of the Helmholtz machine, it might be interesting to examine the computational aspects of a spiking neural network in terms of the difference in description length between individual liquid states, instead of Euclidean distance as was done by Maass.

From the theory and experiments in this project can be concluded that the liquid state machine and the Helmholtz machine are interesting models whose dynamics and capabilities are not yet fully understood. Further research into those types of neural networks might not only give more insight in theoretical and mathematical properties of processing patterns in time series data, but might also shed some light on the working of the brain, and the processes that are involved in the emergence of intelligence.

Bibliography

Atkinson, R. and Shiffrin, R. (1968). Human memory: a proposed system and its control processes. In Spence, K., editor, *Advances in research and theory: The psychology of learning and motivation*, volume 2, pages 89–195. New York Academic Press.

Basili, R., Serafini, A., and Stellato, A. (2004). Classification of musical genre: a machine learning approach. In *Proceedings of the 5th International Conference on Music Information Retrieval*, pages 92–95.

Bayes, T. (1764). An essay toward solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London*, 53:370–418.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Bliss, T. and Lømo, T. (1973). Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path. *Journal of Physiology*, 232:331–356.

Blood, A. and Zatorre, R. (2001). Intensely pleasurable responses to music correlate with activity in brain regions implicated with reward and emotion. In *Proceedings of the National Academy of Sciences*, volume 98, pages 11818–11823.

Boyd, S. and Chua, L. (1985). Fading memory and the problem of approximating nonlinear operators with Volterra series. In *IEEE Transactions on Circuits and Systems*, volume 32, pages 1150–1161.

Buonomano, D. (2000). Decoding temporal information: a model based on short term synaptic plasticity. *Journal of Neuroscience*, 20(3):1129–1141.

Casti, J. (2000). *Five more golden rules: knots, codes, chaos, and other great theories of 20th-century mathematics*, chapter The shannon coding theorem, pages 207–254. John Wiley and Sons, New York.

Chang, J. and Zhang, B. (2001). Using stochastic Helmholtz machine for text learning. In *Proceedings of 19th International Conference on Computer Processing of Oriental Language*, pages 453–458.

Churchland, P. and Sejnowski, T. (1992). *The computational brain.* MIT Press, Cambridge.

Dayan, P. and Hinton, G. (1996). Varieties of Helmholtz machine. *Neural Networks*, 9(8):1385–1403.

Dayan, P., Hinton, G., Neal, R., and Zemel, R. (1995). The Helmholtz machine. *Neural Computation*, 7(5):889–904.

Dennis, J. and Schnabel, R. (1983). *Numerical methods for unconstrained optimization and nonlinear equations.* Prentice-Hall, New York.

Felleman, D. and Essen, D. v. (1991). Distributed hierarchical processing in primate cerebral cortex. *Cerebral Cortex*, 1:1–47.

Feng, J. and Brown, D. (2000). Integrate-and-fire models with nonlinear leakage. *Bulletin of Mathematical Biology*, 62:467–481.

Fernando, C. and Sojakka, S. (2003). Pattern recognition in a bucket. In *Proceedings of the 7th European Conference on Artificial Life*, pages 588–597.

Fishman, Y., Igor, O., Volkov, M., Noh, D., Garell, P., Bakken, H., Arezzo, J., Howard, M., and Steinschneider, M. (2001). Consonance and dissonance of musical chords: neural correlates in auditory cortex of monkeys and humans. *Journal of Neurophysiology*, 86(6):2761–2788.

Gray, P., Krause, B., Atema, J., Payne, R., Krumhansl, C., and Baptista, L. (2001). The music of nature and the nature of music. *Science*, 291:52–54.

Grenander, U. (1976–1981). *Lectures in pattern theory I, II and III: pattern analysis, pattern synthesis and regular structures.* Springer-Verlag, Berlin.

Hagan, M. and Menhaj, M. (1994). Training feedforward networks with the Marquardt algorithm. In *IEEE Transactions on Neural Networks*, volume 5, pages 989–993.

Harford, S. (2003). Automatic segmentation, learning and retrieval of melodies using a self-organizing neural network. In *Proceedings of the 4th International Conference on Music Information Retrieval.*

Häusler, S., Markram, H., and Maass, W. (2003). Perspectives of the high dimensional dynamics of neural microcircuits from the point of view of low dimensional readouts. *Complexity (Special Issue on Complex Adaptive Systems)*, 8(4):39–50.

Hebb, D. (1949). *Organization of behavior.* John Wiley and Sons, New York.

Helmholtz, H. v. (1877). *On the sensations of tone as a physiological basis for the theory of music (translation of the 1877 edition).* Dover Publications, New York.

Hinton, G., Dayan, P., Frey, B., and Neal, R. (1995). The wake-sleep algorithm for unsupervised neural networks. *Science*, 268:1158–1161.

Hochreiter, S. (1998). Recurrent neural net learning and vanishing gradient. *International Journal Of Uncertainity, Fuzziness and Knowledge-Based Systems*, 6(2):107–116.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Kalat, J. (2001). *Biological psychology.* Wadsword, Thomson Learning, Belmont.

Kandel, E., Schwartz, J., and Jessell, T. (1991). *Principles of neural science.* Appleton & Lange, Connecticut.

Kullback, S. (1959). *Information theory and statistics.* John Wiley and Sons, New York.

Li, M. and Sleep, R. (2004). Improving melody classification by discriminant feature extraction and fusion. In *Proceedings of the 5th International Conference on Music Information Retrieval*, pages 44–47.

Lin, T., Horne, B., Tiño, P., and Giles, C. (1996). Learning long-term dependencies is not as difficult with NARX networks. In Touretzky, D., Mozer, M., and Hasselmo, M., editors, *Advances in Neural Information Processing Systems*, volume 8, pages 577–583. MIT Press.

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Computation*, 14:2531–2560.

Maddage, N., Xu, C., and Wang, Y. (2003). An SVM-based classification approach to musical audio. In *Proceedings of the 4th International Conference on Music Information Retrieval.*

McKay, C. and Fujinaga, I. (2004). Automatic genre classification using large high-level musical feature sets. In *Proceedings of the 5th International Conference on Music Information Retrieval*, pages 95–100.

McKinney, M. and Breebaart, J. (2003). Features for audio and music classification. In *Proceedings of the 4th International Conference on Music Information Retrieval*, Baltimore, MD. Johns Hopkins University.

Minsky, M. and Papert, S. (1969). *Perceptrons: an introduction to computational geometry.* MIT Press, Cambridge.

Mumford, D. (1994). *Large-scale neuronal theories of the brain*, chapter Neuronal architectures for pattern-theretic problems. MIT Press, Cambridge.

Mumford, D. and Desolneux, A. (2005). *Pattern theory through examples.* In preparation.

Pantev, C., Hoke, M., Lutkenhoner, B., and Lehnertz, K. (1989). Tonotopic organization of the auditory cortex: pitch versus frequency representation. *Science*, 246:486–488.

Pantev, C., Hoke, M., Lutkenhoner, B., and Lehnertz, K. (1991). Neuromagnetic evidence of functional organization of the auditory cortex in humans. *Acta Otolaryngol Supply*, 491:106–115.

Pearlmutter, B. (1995). Gradient calculations for dynamic recurrent neural networks: a survey. In *IEEE Transactions on Neural Networks*, volume 6, pages 1212–1228.

R.J., W. and Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. In Chauvin, Y. and Rumelhart, D., editors, *Backpropagation: theory, architectures and applications*, pages 433–486. Lawrence Erlbaum Publishers, Hillsdale, N.J.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning internal representations by error propagation. *Parallel Distributed Processing*, 1:318–362.

Savage, J. (1998). *Models of computation: exploring the power of computing.* Addison-Wesley, Boston.

Thompson, C. (1988). *Classical equilibrium Statistical Mechanics.* Clarendon Press, Oxford.

Thorpe, S., Delorme, A., and Rullen, R. v. (2001). Spike-based strategies for rapid processing. *Neural Networks*, 14:715–725.

Vapnik, V. (1998). *Statistical learning theory.* John Wiley and Sons, New York.

Vreeken, J. (2004). On real-world temporal pattern recognition using liquid state machines. Master's thesis, Utrecht University, University of Zurich.

Weinberger, N. (1994a). A few notes on pitch. *MuSICA Research Notes*, 1(1).

Weinberger, N. (1994b). Musical building blocks in the brain. *MuSICA Research Notes*, 1(2).

Whitfield, I. (1980). Auditory cortex and the pitch of complex tones. *Journal of the Acoustical Society of America*, 67:644–647.

Williams, R. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280.

Zatorre, R. (2005). Neuroscience: finding the missing fundamental. *Nature*, 436:1093–1094.

Zemel, R. and Hinton, G. (1994). Developing population codes by minimizing description length. In D.Cowan, J., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems*, volume 6, pages 3–10. Morgan Kaufmann Publishers, Inc.