Supervised Learning in Multilayer Spiking Neural Networks

Ioana Sporea
Department of Computing
University of Surrey

A thesis submitted for the degree of

Doctor of Philosophy

March 2012
Abstract

In this thesis, a new supervised learning algorithm for multilayer spiking neural networks is presented. Gradient descent learning algorithms have led traditional neural networks with multiple layers to be one of the most powerful and flexible computational models derived from artificial neural networks. However, more recent experimental evidence suggests that biological neural systems use the exact time of single action potentials to encode information. These findings have led to a new way of simulating neural networks based on temporal encoding with single spikes. Analytical demonstrations show that these types of neural networks are computationally more powerful than networks of rate neurons.

Conversely, the existing learning algorithms no longer apply to spiking neural networks. Supervised learning algorithms based on gradient descent, such as SpikeProp and its extensions, have been developed for spiking neural networks with multiple layers, but these are limited to a specific model of neurons, with only the first spike being considered. Another learning algorithm, ReSuMe, for single layer networks is based on spike-timing dependent plasticity (STDP) and uses the computational power of multiple spikes; moreover, this algorithm is not limited to a specific neuron model.

The algorithm presented here is based on the gradient descent method, while making use of STDP and can be applied to networks with multiple layers. Furthermore, the algorithm is not limited to neurons firing single spikes or specific neuron models. Results on classic benchmarks, such as the XOR problem and the Iris data set, show that the algorithm is capable of non-linear transformations. Complex classification
tasks have also been applied with fast convergence times. The results of the simulations show that the new learning rule is as efficient as SpikeProp while having all the advantages of STDP. The supervised learning algorithm for spiking neurons is compared with the back-propagation algorithm for rate neurons by modelling an audio-visual perceptual illusion, the McGurk effect.
Contents

Contents iii

List of Figures viii

1 Introduction 1

2 The Biological Nervous System 4
   2.1 The structure of the neuron 4
   2.2 Electrical signalling 7
   2.3 Synapses 10
   2.4 Synaptic plasticity 11

3 Rate-Coded Neural Networks 13
   3.1 Networks of neurons 13
      3.1.1 McCulloch-Pitts neurons 14
      3.1.2 The perceptron 15
      3.1.3 Perceptron learning 17
   3.2 Multilayer networks and gradient descent learning 18
3.2.1 The network architecture .................................................. 19
3.2.2 The back-propagation algorithm ........................................... 20

4 Spiking Neuron Models .......................................................... 27
4.1 Hodgkin-Huxley model ........................................................ 28
4.2 Simple models ................................................................. 32
  4.2.1 The leaky Integrate-and-fire model ................................. 32
  4.2.2 Spike Response Model (SRM) ........................................... 34
4.3 Discussion ................................................................. 35

5 Learning in Spiking Neural Networks ....................................... 37
5.1 Spike-Time Dependent Plasticity (STDP) .................................. 37
5.2 Supervised learning methods ................................................. 39
  5.2.1 The tempotron ............................................................ 39
  5.2.2 ReSuMe - Remote Supervision ...................................... 41
  5.2.3 SpikeProp - Gradient based learning ................................. 43
  5.2.4 Other supervised learning methods ................................... 48

6 Analysis of SpikeProp .......................................................... 55
6.1 SpikeProp convergence ........................................................ 56
  6.1.1 Experimental setup .................................................... 56
  6.1.2 Results ................................................................. 58
6.2 Reference start time .......................................................... 62
6.3 Discussion ................................................................. 66
# List of Figures

2.1 Nerve cell in the motor cortex. Reproduction of an ink drawing by Ramón y Cajal (1909) showing the diversity of neurons forms and sizes in the motor cortex. ........................................... 5

2.2 A typical nerve cell with soma, dendrites and axon. Reproduced from Beatty (2001), Figure 2.7. ........................................................ 6

2.3 Example of an action potential on different time scales. Reproduced from Beatty (2001), Figure 3.5. ............................................. 8

2.4 Examples of patterns of action potentials. A. Suppressed firing; B. Clocklike regular spikes; C. Repetitive burst firing; D. Complex spiking pattern. Reproduced from Beatty (2001), Figure 3.6. ... 9

3.1 Examples of logical functions implemented with McCulloch-Pitts units, where inhibitory connections are marked with a small circle attached to the end of the edge. The threshold of the unit is shown inside the large circle. ......................................................... 14

3.2 A typical perceptron architecture. ................................................ 16

3.3 Patterns a and b are fully connected, and patterns c and d are not connected. ................................................................. 16

3.4 Separation of input space with a perceptron. .............................. 17

3.5 A generic multilayer network. ..................................................... 19
LIST OF FIGURES

3.6 The sigmoid function for $c = 1$. ................................................. 21

3.7 Back-propagation: (a) without; (b) with momentum. Reproduced after Rojas (1996), Figure 8.1. ................................................. 25

4.1 Schematic diagram for the Hodgkin-Huxley model. Reproduced after Gerstner and Kistler (2002), Figure 2.2. ............................ 29

4.2 Schematic diagram of the integrate-and-fire model. ....................... 32

6.1 The percentage of successful learning trials of XOR problem for different combinations of learning rates and thresholds. ............... 59

6.2 Average number of iterations (calculated over the successful trials) needed for convergence for the XOR problem for different combinations of learning rates and thresholds: (a) without, (b) with a momentum parameter ......................................................... 60

6.3 Standard deviation of the number of iterations needed for convergence for the XOR problem for different combinations of learning rates and thresholds. ................................................. 61

6.4 The percentage of successful learning trials of XOR problem with negative and positive initial weights within the range (-0.1, 0.9). 62

8.1 Analysis of the learning process for the XOR benchmark with the parameters $A_+ = 1.2$ and $A_- = 0.5$: (a) The network error during learning. (b) The Euclidean distance between the weight vector solution and the weight vectors during the learning process. (c) The output signals during learning for each of the four patterns. The x markers represent the target spike times. (d) The hidden signals during learning for each hidden neuron for one input pattern $(0 \ 0)$. ................................. 93
8.2 Output signals during learning for the Iris data set for all input patterns for a sample trial. The x markers represent the target spike times. ................................................. 95

8.3 (a) Network structure for the XOR problem. A feed-forward network with three layers, where the input layer consist of two groups of 20 neurons for each logical signal. (b) Each logical signal is encoded by a set of 20 spike trains, where the input signals are represented by spike trains with at least one spike, and the output signals are represented by spike trains with exactly three spikes. Reproduced after Grüning and Sporea (2012). .................. 97

8.4 Output signals for all input patterns during learning for a sample trial. The x markers represent the target spike trains. .............. 99

8.5 The input, hidden, and output signals after the learning process has converged. The first 20 input spike trains represent the logical symbol "1", while the other 20 input spike trains represent the logical symbol "0". The grey signals in the output graph represent the target pattern. ................................. 100

8.6 The output signals during learning for a sample trial. The network has been trained with 6 non-noisy patterns that span over 100 ms. The x markers represent the target spike trains. .......... 101

8.7 The accuracy on noisy patterns: The network has been trained with 10 non-noisy patterns that span over 100 ms. The random performance level is in this case 10%. The error bars show the standard error of the mean. ............................. 104

8.8 Example of an input pattern where the spikes are moved by a time interval within a Gaussian distribution with mean 0 and standard deviation 4 ms. The grey markers represent the original signals, while the black markers represent the noisy signals. .......... 105
8.9 The output signals during learning for a sample trial. The network has been trained with 3 noisy patterns (4 ms jitter) that span over 500 ms. The x markers represent the target spike trains. 106

8.10 The accuracy on noisy patterns: The network has been trained with 3 noisy patterns that span over 500 ms. During learning the noisy input patterns are generated by moving each spike within a Gaussian distribution with mean 0 and standard deviation 4 ms. The random performance level is in this case 33%. The error bars show the standard error of the mean. 107

9.1 Feed-forward network model of the McGurk effect. (a) Late integration model. (b) Early integration model. Reproduced from Sporea and Grüning (2010). 118

9.2 Feed-forward spiking neural network model of the McGurk effect. 124
List of Publications


Chapter 1

Introduction

Gradient descent learning algorithms have led sigmoidal neural networks to be one of the most powerful and flexible computational model derived from artificial neural networks. These neural networks are capable of self-organization and are able to approximate arbitrarily well any continuous function with a compact domain and range (Rojas, 1996). Traditional views of the artificial neurons consist of representing an analog variable through the firing rate of a neuron (Maass, 1997a,b). That is, the output of a sigmoidal unit is interpreted as a representation of the firing rate of the biological neuron.

Findings which suggest that neural systems use the exact time of single action potentials to encode information (Johansson and Birznieks, 2004; Thorpe and Imbert, 1989) have led to a new way of simulating neural networks based on temporal encoding with single spikes (Maass, 1997b). Investigations of the computational power of spiking neurons have shown that spiking neurons can arbitrarily approximate any continuous function and are computationally more powerful than sigmoidal neurons (Maass, 1997c). These theoretical investigations have not yet been confirmed by experimental results. One reason for this lack of experimental results that would prove the computational superiority of spiking neurons over rate neurons is the need for a general-purpose supervised learning algorithm.
1. Introduction

Models of spiking neurons have been developed with different degrees of realism, from compartmental models that are biologically plausible, but computationally expensive to simulate, to simple phenomenological models that can be easily simulated. In order to realise the computational power of spiking neurons, an efficient learning method must be developed. Experimental evidence has shown that instruction-based learning is present in the brain, especially in sensorimotor networks (Knudsen, 1994). Another example of supervised learning in the brain is in sensory systems. Experiments have shown that auditory orienting behaviour in barn owls is a result of adaptive adjustments of sound localization by instructive signals (Knudsen, 2002). Supervised learning is also likely to contribute to certain cognitive skills, such as pattern recognition or language acquisition, however without experimental confirmation (Knudsen, 1994).

The current thesis explores the available supervised learning algorithms in rate-coded and spiking neural networks. Gradient descent based algorithms are one of the most used learning methods for rate neurons. The back-propagation version for spiking neural networks, SpikeProp has been developed by Bohte et al. (2002). Although it has been reported that SpikeProp requires less learning iterations than the classic back-propagation, a closer inspection shows that SpikeProp iterations involve more computations than back-propagation. Systematic investigations on the learning algorithm for spiking neurons also reveal that the network error surface has sudden accelerations that significantly delay the learning process (Fujita et al., 2008; Takase et al., 2009). Other studies performed with SpikeProp show that mixed sign weights allow the algorithm to converge (Moore, 2002; Takase et al., 2009), although it was previously considered impossible. Other misconceptions about SpikeProp include the role of the additional input neuron that designates the reference start time (Sporea and Grünig, 2011).

In this thesis, a new supervised learning algorithm, multilayer ReSuMe (Sporea and Grünig, 2012), for feed-forward networks of spiking neurons with multiple layers is introduced. The learning rule extends the ReSuMe algorithm (Ponulak and Kasinski, 2010) to multiple layers using backpropagation of the network error. The weights are updated according to STDP and anti-STDP processes and unlike SpikeProp (Bohte et al., 2002), multilayer ReSuMe can be applied to neuron...
rons firing multiple spikes in all layers. Multilayer ReSuMe is analogous to the backpropagation learning algorithm for rate neurons, while making use of spiking neurons. To the best of our knowledge this is the first fully supervised learning algorithm for spiking neuron networks with hidden layers where multiple spikes are considered in all layers. The learning rule is tested with various benchmarks, such as the XOR problem and the Iris data set (Fisher, 1936), as well as complex classification problems with randomly generated patterns with and without noise.

A case study in the form of modelling a speech perception illusion, the McGurk effect (McGurk and MacDonald, 1976), is also described. The McGurk effect is modelled with two rate-coded neural networks trained with back-propagation. The models simulate two theories regarding the cause of the effect. The phenomenon is also modelled with a feed-forward network of spiking neurons trained with multilayer ReSuMe and the results are compared.

The rest of this thesis is organised as follows: The second chapter contains a detailed description of biological neurons. In the third chapter a brief introduction to the dynamic research field of rate-coded neural networks is given, starting with the first models of artificial neurons to one of the most used network architectures, sigmoidal neurons. The third generation of neural networks is briefly examined in the fourth chapter. Some of the most popular spiking neuron models are described in Chapter 4, together with existing supervised and unsupervised learning algorithms for spiking neural networks in Chapter 5. One of the existing algorithms for multilayer spiking neural networks based on gradient descent techniques is critically examined in Chapter 6. The proposed learning rule for spiking neurons is derived in Chapter 7 along with an analysis of weight modifications for a simplified network with a single output neuron. In Chapter 8 the flexibility and power of the feed-forward spiking neural networks trained with multilayer ReSuMe are showcased by non-linear problems and classifications tasks. The spiking neural network is trained with spike timing patterns distributed over timescales in the range of tens to hundreds of milliseconds, comparable to the span of sensory and motor processing (Mauk and Buonomano, 2004). The McGurk effect models are described and analysed in Chapter 9. The report ends with a concluding chapter where future research directions are discussed.
Chapter 2

The Biological Nervous System

The functions of the nervous system underlie our perception of the external world and control our behaviour. Our ability to exploit the physical environment is achieved by means of a complex system of sensory receptors connected to the brain. Using nerve cells and the connections between them, the brain organises the information from the receptors into perceptions and then into appropriate actions (Kandel et al., 2000).

The nervous system consists of two main classes of cells: nerve cells (neurons) and glial cells (Kandel et al., 2000). While the glial cells are not known to be directly involved in the informational process, providing only mechanical and metabolic support, the neurons are the cells with information processing functions. Ramón y Cajal (1909), one of the pioneers of neuroscience in late nineteenth century, gathered the first evidence that individual cells are the basic elements of the nervous system. The diversity of shapes and sizes of neurons in the nervous system is shown in Figure 2.1 in a drawing by Ramón y Cajal (1909).

2.1 The structure of the neuron

Despite the variety of forms and sizes, all neurons share common features in their structure. A typical neuron can be divided into four distinct regions: the
Figure 2.1: Nerve cell in the motor cortex. Reproduction of an ink drawing by Ramón y Cajal (1909) showing the diversity of neurons forms and sizes in the motor cortex.
2. The Biological Nervous System

cell body or soma, dendrites, axon, and the presynaptic terminals (see Figure 2.2). The soma contains the nucleus of the cell and intracellular structures and is similar to any other cell in the body (Beatty, 2001). The cell body give rise to two types of specialised extensions, namely several dendrites and one axon; these extensions are found only in nerve cells.

Dendrites are processes that receive incoming signals from other cells and brings it to the cell body of the neuron. The axon carries the electrical signal from the cell body to other neurons or muscle cells. Axons terminate in presynaptic terminals, which are the information transmitters. Most presynaptic terminals end on the postsynaptic neuron's dendrites or its cell body, but connections between axons can also occur (Kandel et al., 2000).

![Diagram of a nerve cell with soma, dendrites, and axon]({attachment:diagram.png})

Figure 2.2: A typical nerve cell with soma, dendrites and axon. Reproduced from Beatty (2001), Figure 2.7.

The point of communication between two neurons is called synapse. Synapses have directional functions, transmitting the information from the presynaptic terminal of the sending cell, called presynaptic cell, and affecting the behaviour of the receiving cell, called postsynaptic cell (Kandel et al., 2000). The cell body integrates the input received from the synaptic connections and determines the
message to be transmitted by the axon to other cells. In the human brain, a single cell may receive input from as many as 100,000 individual synapses from other cells (Beatty, 2001).

2.2 Electrical signalling

The nerve cells make sense of the world around us by perceiving, making decisions and evoking behaviour. The diversity and complexity of mental functions of the nervous system is the result of simple cellular processes that occur in groups of individual neurons (Beatty, 2001).

Neurons use electrical potentials, called action potentials, to signal and process information by means of ions of sodium, potassium, and other elements. Electrical signals within living cells are carried by the movements of ions across the membrane of cells; ions can cross the membrane through ion channels. Although over a hundred different types of ion channels exist in various types of living cells, potassium, sodium, chloride, and calcium channels are of particular importance in neural signalling and communication (Beatty, 2001).

The electrical properties of the cell membrane determine the signals used by nerve cells. The membrane potential of a nerve cell is the voltage difference between the interior and the exterior of the cell at a given moment. The resting potential of a cell is the membrane potential in the absence of electrical signalling. Using giant squid neurons and assuming they are similar to those of mammalian neurons, Hodgkin and Huxley experimentally measured the resting potential to be about -70 mV. In different nerve cells, the resting potential can range from -40 to -80 mV. They also demonstrated that the resting membrane is primarily permeable to potassium, but also slightly permeable to sodium (Beatty, 2001).

The action potential is the standard signal used to transmit information between nerve cells. Because of its sharp change in the membrane voltage, it is also referred to as nerve impulse or spike. Figure 2.3 shows an example of an action potential. The action potential is usually triggered in the axon hillock (the initial
2. The Biological Nervous System

segment of the axon) at the junction between the cell body and the axon. The electrical potential is then conducted along the axon to its target with a constant amplitude of around 100 mV (Kandel et al., 2000). In order to assure its reliability in carrying information, the action potential is continuously regenerated by the membrane of the axon from the initial segment of the axon to its end terminals (Beatty, 2001).

Because of the way action potentials are generated at the membrane of the cell, action potentials are all-or-none events. Also, all action potentials or spikes have about the same amplitude and duration, and wave form. Thus, neurons transmit information by the number of spikes and the time intervals between them, not by the size and shape of the action potentials (see Figure 2.4).

Neurons transmit information by varying the rate and the pattern of the action potentials (Beatty, 2001). As the form of the spike remains the same, messages are transmitted using different patterns of spikes, as shown in Figure 2.4. When the axon is stimulated by a weak current, the axon becomes slightly less negative. Because the membrane becomes less charged or less polarised, this process
is termed depolarisation. Conversely, the process of decreasing the membrane potential is termed hyperpolarisation. Hyperpolarisation makes a cell less likely to trigger a spike and is therefore called inhibitory. If the stimulating current is sufficiently strong, a larger depolarisation may produce an action potential. A spike is elicited if the depolarisation induced by the stimulus reaches the neuron's characteristic threshold. The threshold is usually 5 to 10 mV higher than the resting potential of the nerve cell, so if the resting potential is -70 mV, the threshold will be between -65 to -60 mV (Beatty, 2001). When a nerve impulse is elicited, the membrane potential becomes momentarily positive, the squid membrane potential reaching values of about +40 mV (Beatty, 2001).

![Figure 2.4: Examples of patterns of action potentials. A. Suppressed firing; B. Clocklike regular spikes; C. Repetitive burst firing; D. Complex spiking pattern. Reproduced from Beatty (2001), Figure 3.6.](image)

At the end of a pulse, the membrane potential does not return directly to the resting potential, but actually goes below the resting potential. This hyperpo-
2. The Biological Nervous System

Polarization of the membrane is called spike after-potential (Gerstner and Kistler, 2002). Due to this hyperpolarization, the membrane is unable to produce two nerve impulses in rapid succession. The period of time after a spike was elicited and no other spike can be initiated is termed absolute refractory period and it is about a millisecond for most neurons (Beatty, 2001). Following the absolute refractory period, for a somewhat longer period, there is a temporary hyperpolarization of the neuron and a raise of the threshold. This period of heightened threshold is called relative refractory period, when a stronger input current is required to generate another action potential.

2.3 Synapses

Synapses are points of communication between the axon terminal of one neuron and the membrane of another. Synapses can be either chemical or electrical. Electrical synaptic transmission is instantaneous and can be bidirectional, while chemical synapses can amplify neural signals and can produce more complex behaviour. Chemical synapses use neurotransmitters for synaptic transmission, which are responsible for a delay of at least 0.3 ms (usually 1-5 ms or longer) (Kandel et al., 2000).

Depending on the microscopic structure, synapses also have different functional properties of synaptic communication in the nervous system. One classification of synapses is according to the effect on the postsynaptic cell, as synapses can be either excitatory or inhibitory (Beatty, 2001).

When the neurotransmitter is released at an excitatory synapse, the response is to move the membrane potential of the postsynaptic cell towards the threshold for producing a spike. Such a response is called excitatory postsynaptic potential (EPSP). A single EPSP depolarises the postsynaptic membrane by less than 1 mV (usually 0.2-0.4 mV), not enough to generate an action potential.

Conversely, input from an inhibitory synapse decreases the probability that the neuron will fire. This response is called inhibitory postsynaptic potential.
(IPSP). While a single EPSP is not large enough to trigger a spike, an IPSP can prevent an action potential, if it is strong enough to counteract the sum of excitatory signals (Kandel et al., 2000). Although they have opposing effects, both postsynaptic potentials have similar features. As excitation and inhibition have opposite effects, the postsynaptic potential is the result of the balance between excitatory and inhibitory influences (Beatty, 2001).

In most neurons, the effect produced by a single synapse is usually insufficient to trigger an action potential. In order to produce a spike, the membrane potential integrates the inhibitory and excitatory signals into a single response.

2.4 Synaptic plasticity

The precise synaptic connections between nerve cells are mainly responsible for the behavioural actions performed by the brain. Neural circuits are fine-tuned by interactions between the organism and its environment. These modifications of the synaptic connections represent the physiological basis of learning (Kandel et al., 2000). At the cellular level, the process of adapting to the environment through changes in the behavioural responses is called plasticity.

One of the simplest forms of learning is habituation – learning about the properties of a new harmless stimulus that is neither rewarded nor punished (Beatty, 2001). Experimental studies have shown that repeated stimulation leads to a decrease in the strength of the synaptic connection between excitatory interneurons and motor neurons (Beatty, 2001). Another form of learning is sensitisation – when a harmful stimulus is presented, the animal learns to respond more vigorously (Beatty, 2001). This leads to an enhancement of synaptic efficiency between sensory neurons and interneurons.

More complex forms of learning require storing information about places, objects or people and imply a conscious recall. Long-term potentiation (LTP) and long-term depression (LTD) are two distinct types of changes in the synaptic efficiency that underlie explicit forms of learning (Kandel et al., 2000). Long-term
potentiation results in a stronger effect of a presynaptic input in a postsynaptic cell as a consequence of using that synapse. Long-term depression is the opposite effect and results in a decrease of synaptic strength (Beatty, 2001).

Studies performed on intact animals and in cell cultures have shown that long-term potentiation can be associative or non-associative. Non-associative changes in the synaptic strength are determined only by presynaptic activity. Associative LTP changes require concomitant activity in both the presynaptic and postsynaptic cells. These findings on associative LTP provide direct evidence for Hebb's postulate: synaptic connections are modified when pre- and postsynaptic nerve cells are synchronously active (Hebb, 1949).
Chapter 3

Rate-Coded Neural Networks

The first model of artificial neural networks has been introduced more than half a century ago by Warren McCulloch and Walter Pitts (1943) by describing the behaviour of neural networks in terms of propositional logic. New models of networks and learning algorithms that attempt to mimic the information processing capabilities of the nervous system have been proposed ever since.

This chapter gives a brief introduction to the broad and dynamic research field of rate-coded neural networks. Some of the most important models of rate-coded neural networks and some of the earlier models (threshold units and perceptrons) are described along with one of the most used learning algorithms, the back-propagation method.

3.1 Networks of neurons

Due to the "all-or-none" character of the action potential of nerve cells, McCulloch and Pitts described the first abstract neuron in terms of propositional logic (McCulloch and Pitts, 1943). Thus, binary values are associated with triggering an action potential or remaining quiescent. Considering the general definition of an artificial neural network as a network of primitive functions (Rojas, 1996), the nodes of the network are also called computing elements or units.
3.1.1 McCulloch-Pitts neurons

The McCulloch-Pitts units use only binary signals and the nodes produce only binary results. The network is composed of directed unweighted edges which can be either excitatory or inhibitory. McCulloch-Pitts units use absolute inhibition, meaning that if at least one of the inhibitory signals is 1 then the result of the computation is 0. If all inhibitory signals are null, the units have an associated threshold used to compare the total excitation and to produce the output, with the step function as the activation function.

The McCulloch-Pitts networks act as threshold gates capable of implementing any given logical function of \( n \) arguments (Rojas, 1996). While simple logical function, such as AND, OR, and NOT, can be implemented directly with a single McCulloch-Pitts unit, any other logical function can be computed with a network of two or more layers by combining units which exclusively compute that particular function (Rojas, 1996). Figure 3.1 shows several examples of logical functions implemented with McCulloch-Pitts neurons; in the diagrams the inhibitory connections are marked with a small circle attached to the end of the edge.

![Diagram of logical functions implemented with McCulloch-Pitts neurons.](image)

Figure 3.1: Examples of logical functions implemented with McCulloch-Pitts units, where inhibitory connections are marked with a small circle attached to the end of the edge. The threshold of the unit is shown inside the large circle.
It has been demonstrated that networks using units with more general properties, such as weighted edges and relative inhibition (the effect of inhibitory edges is to increase the firing threshold), are equivalent to networks with McCulloch-Pitts units, meaning that weighted networks can achieve the same results but with fewer computing elements (Rojas, 1996). The main difference between the two classes of models is the type of learning that can be used. While in McCulloch-Pitts networks only the threshold and the connectivity can be adapted, in weighted networks learning consists of finding an optimal combination of weights and usually keeping the topology of the network fixed.

3.1.2 The perceptron

Numerical weights were first introduced in 1958 by Frank Rosenblatt, an American psychologist, who proposed a more general computation model than McCulloch-Pitts networks, called the perceptron (Rosenblatt, 1958). The model was later refined and perfected by Minsky and Papert in the 1960's (Minsky and Papert, 1969). Minsky and Papert analysed the computational properties of the perceptron under different assumptions. The model used is a retina of pixels with binary values, where the threshold units may be considered as visual feature detectors. The units are connected to a single output threshold unit (see Figure 3.2). The system is trained to recognise certain input patterns, by searching a set of suitable connection weights.

Minsky and Papert's analysis of this architecture revealed fundamental limitations of the perceptrons. The main limitation of this type of system is concerned with the detection of certain global properties of the input patterns. One such example is the connectedness of a geometric figure; that is, a perceptron is not capable of learning to distinguish between two patterns that are fully connected (a and b in Figure 3.3) from two patterns that are not connected (c and d in Figure 3.3). Similar findings apply to the determination of parity of the number of active input units (Minsky and Papert, 1969).

These demonstrations are extremely important as they state that global prop-
3. Rate-coded Neural Networks

Feature units

Output unit

Figure 3.2: A typical perceptron architecture.

Figure 3.3: Patterns a and b are fully connected, and patterns c and d are not connected.

Properties cannot be learnt by any local computations. In order to make a decision about global properties, such as the connectedness of a geometric figure or the parity of the number of active units, perceptrons that have access to the whole figure would be necessary in this kind of system (Rojas, 1996).

In determining the capabilities of perceptron networks another question is what classes of problems can be computed by a single perceptron. Taking a two-dimensional vector space as an example (see Figure 3.4), a perceptron can separate classes that lie on either side of a straight line. However, there are classifications that are not linearly separable and cannot be computed by the perceptron (see Chapter 3 in Rojas (1996) for a complete demonstration). One such example is the exclusive-or (XOR) problem.

The requirement of linear separability is general for any n-dimensional vector space: a perceptron may only classify regions that can be divided by an \((n - 1)\)-dimensional hyperplane (Ellis and Humphreys, 1999).
Global properties, such as the parity function, can still be detected by simple perceptrons if the input feature units would depend on all points in the investigated space. As shown by Minsky and Papert (1969) linear separation depends on the specific encoding of the information. Thus, single layer perceptrons could solve linearly inseparably problems by encoding information in non-standard ways.

3.1.3 Perceptron learning

The method by which a neural network adapts its parameters so that the network will respond in a particular way is called learning. The learning algorithm implies a correction step that is executed iteratively until the network has learnt to produce the desired output (Rojas, 1996).

There are two main classes of learning methods: supervised and unsupervised learning (Rojas, 1996). Supervised learning algorithms require a set of inputs and target outputs to be collected and presented to the network. The output computed by the network is compared with the desired response. The weights are then updated according to the magnitude of the deviation between the actual output and the target output. Unsupervised methods apply when the exact output is unknown. In the following, an unsupervised learning rule is presented, Hebb’s postulate, as well as a supervised learning algorithm for perceptrons, the Widrow-Hoff delta rule.
3. Rate-coded Neural Networks

Hebbian learning

Hebb’s postulate states that the synaptic connection between two neurons is strengthened if they are simultaneously active (Hebb, 1949). Formally, in Hebbian learning the weight $w_{ji}$ between two units $i$ and $j$ is changed according to:

$$\Delta w_{ji} = \gamma a_i a_j$$  \hspace{1cm} (3.1)

where $a_i$ and $a_j$ are the activation values in units $i$ and $j$ respectively. The factor $\gamma$ is the learning rate that determines the step length of the weight change. The updated set of weights is the correlation matrix between the input and the output vector.

The Widrow-Hoff delta rule

The delta rule developed by Widrow and Hoff is one of the most used supervised learning rules for single layer perceptrons (Beale and Jackson, 1990). For each given input sample, the computed output is compared with the target pattern and the weights are modified in order to reduce the difference between the two patterns:

$$\Delta w_{ji} = \gamma a_i (t_j - a_j)$$  \hspace{1cm} (3.2)

where $a_i$ and $a_j$ are the activation values in units $i$ and $j$ respectively, and $t_j$ is the target pattern for unit $j$. Again, the factor $\gamma$ represents the learning rate. It has been shown that in case of a linear activation function, the delta rule is able to find the optimal set of weights for single layer perceptrons.

3.2 Multilayer networks and gradient descent learning

The limitation of linear separability of the perceptron was overcome by Rumelhart and McClelland (1986) when they proposed their improved model, the multilayer
3. Rate-coded Neural Networks

perceptron. In order to increase the computational power of the network the threshold units are structured in successive layers of computing elements. In contrast to the threshold units and the perceptrons, these units apply an activation function that has a continuous output and as such the output is interpreted as the firing rate of a neuron. As a consequence, rate neurons can compute functions with analog input and output.

3.2.1 The network architecture

The set of input units is called the input layer and the set of output units is called the output layer. The input units are just setting the information into the network, without performing any computation. The output is read off from the output units. All other layers that do not have direct connections from or to the outside are called hidden layers. When the units are connected only in one direction, from the input units to the output units, the system is called a feedforward network. In a layered network usually all units in one layer are connected to all units in the following layer. A general structure of the feed-forward network is shown in Figure 3.5.

![Figure 3.5: A generic multilayer network.](image)
3. Rate-coded Neural Networks

3.2.2 The back-propagation algorithm

Although multilayer networks have more computational power than networks with a single layer, the necessary topologies are more complicated and with more parameters, increasing the computational effort needed for finding the optimal combination of weights. The back-propagation algorithm is one of the most studied and most used learning algorithms for neural networks. It has been developed independently by several people (le Cun, 1985; Parker, 1985; Rumelhart et al., 1986).

The back-propagation algorithm is a generalisation of the delta rule and is using the method of gradient descent to minimise the error function in the weight space. The solution of the learning problem is considered to be the set of weights that minimises the error function. As this method requires the computation of the gradient of the error function, the activation function must be continuous and differentiable in order for the composite function and the error function to be continuous as well. One of the most popular activation functions used with back-propagation networks is the sigmoid, which is defined by the expression:

\[ s(x) = \frac{1}{1 + \exp(-cx)} \]  

where \( x \) is a real number, and the coefficient \( c \) is a real number constant. The sigmoid output range contains all numbers between 0 and 1. The extreme values can be reached only asymptotically. The greater the constant \( c \) is, the closer the shape of the function is to that of the step function. The shape of the sigmoid function with the constant \( c = 1 \) is shown in Figure 3.6. In the following expressions the sigmoid function will be used with the coefficient \( c = 1 \).

The derivative of the sigmoid function with the coefficient \( c = 1 \) with respect to \( x \), is

\[ \frac{d}{dx} s(x) = \frac{\exp(-x)}{(1 + \exp(-x))^2} = s(x)(1 - s(x)) \]  

The output of a sigmoidal unit for the input vector \((x_1, ..., x_n)\) and the weight vector \((w_1, ..., w_n)\) is
3. Rate-coded Neural Networks

Figure 3.6: The sigmoid function for $c = 1$. 

$$o_i = \frac{1}{1 + \exp(-\sum_j^n w_j x_j)}$$  \hspace{1cm} (3.5)

The learning problem consists of finding the optimal combination of weights in order for the neural network to approximate a given function as close as possible. The network function is defined only implicitly through a set of sample patterns $(x_1, t_1), \ldots, (x_p, t_p)$ consisting of $p$ ordered pairs of vectors, called the input and target output patterns.

When the input pattern $x_i$ is presented to the network, the output $o_i$ is usually different from the expected output pattern $t_i$. The purpose of the back propagation algorithm is to make $t_i$ and $o_i$ identical for $i = 1, 2, \ldots, p$. This is done by minimising the error function of the network, defined as follows:

$$E = \frac{1}{2} \sum_i^p ||o_i - t_i||^2$$  \hspace{1cm} (3.6)
3. Rate-coded Neural Networks

When the error function reaches an acceptable value for the training set, the network is expected to interpolate any new patterns that are presented. The neural network must decide if the input is similar to any of the learnt patterns and produce an appropriate output. Because the network error $E$ is calculated through composition of the node functions, it is a continuous and differentiable function of all $l$ weights $w_1, w_2, ..., w_l$ of the network. The back-propagation algorithm finds a local minimum for the error function by computing its gradient:

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_l}\right)$$  \hspace{1cm} (3.7)

The weights are corrected using the increment:

$$\Delta w_i = -\gamma \frac{\partial E}{\partial w_i}$$  \hspace{1cm} (3.8)

where $\gamma$ is a learning constant, which defines the step length of weight changes during each iteration in the negative gradient direction.

The algorithm can be decomposed in the following four steps (as described in Rojas (1996)):

1. Feed-forward computation.
2. Weight modifications for the output neurons.
3. Weight modifications for the hidden neurons.
4. Weight updates.

The algorithm will be further described for a feed-forward network with a single hidden layer for simplicity. The formulae can then be easily generalized for networks with any number of layers. A neural network with $n$ input units, $k$ hidden units, and $m$ output units will be considered. The weight matrix between the input layer and the hidden layer will be called $W_1$, with the component $w_{ji}^{(1)}$ as the weight between the input unit $i$ and hidden unit $j$. Similarly, the weight matrix between the hidden layer and the output layer will be called $W_2$, with the component $w_{ji}^{(2)}$ as the weight between the hidden unit $i$ and the output unit $j$. 

22
First step: Feed-forward computation

The input vector \( x = (x_1, ..., x_n) \) is presented to the network and the output for each layer is computed. The vector \( o^{(1)} \) whose components are the outputs of the hidden layer is given by the expression:

\[
o^{(1)} = s(xW_1)
\]  

(3.9)

by applying the sigmoid function to each element of the argument vector. The output of the network will be the \( m \)-dimensional vector given by the expression:

\[
o^{(2)} = s(o^{(1)}W_2)
\]  

(3.10)

The derivatives of the activation functions are also computed and stored at each computing unit.

Second step: Weight modifications for the output neurons

The first set of partial derivatives to be computed is \( \frac{\partial E}{\partial w_{ji}^{(2)}} \), where \( w_{ji}^{(2)} \) is the weight between the hidden unit \( i \) and the output unit \( j \). The information from this weight ending in the output unit \( j \) of the network in the feed-forward step is \( o^{(1)}w_{ji}^{(2)} \), where \( o^{(1)} \) is the stored output of hidden unit \( i \). The back-propagation step computes the gradient of the network error \( E \) with respect to this input, considering \( o^{(1)} \) a constant:

\[
\frac{\partial E}{\partial w_{ji}^{(2)}} = o^{(1)}_i \frac{\partial E}{\partial o^{(1)}_i w_{ji}^{(2)}}
\]  

(3.11)

From equations 3.6 and 3.10 we can calculate the partial derivative of the error function as follows:

\[
\frac{\partial E}{\partial o^{(1)}_i w_{ji}^{(2)}} = o^{(2)}_j \left( 1 - o^{(2)}_j \right) \left( o^{(2)}_j - t_j \right)
\]  

(3.12)

If the back-propagated error of the output node \( j \) is \( \delta^{(2)}_j \), the partial derivatives
of the network error $E$ with respect to $w_{ji}^{(2)}$ will be:

$$\frac{\partial E}{\partial w_{ji}^{(2)}} = o_i^{(1)} \delta_j^{(2)} \quad (3.13)$$

The back-propagated error of output unit $j$ is computed as follows:

$$\delta_j^{(2)} := o_j^{(2)} \left( 1 - o_j^{(2)} \right) \left( o_j^{(2)} - t_j \right) \quad (3.14)$$

**Third step: Back-propagation to the hidden layer**

The partial derivatives $\frac{\partial E}{\partial w_{ji}^{(1)}}$ will be computed, with $w_{ji}^{(1)}$ the weight between the input unit $i$ and the hidden unit $j$, which is connected to each output unit $l$ with a weight $w_{lj}^{(2)}$. The back-propagated error is computed by considering all possible backward paths:

$$\delta_j^{(1)} := o_j^{(1)} \left( 1 - o_j^{(1)} \right) \sum_l w_{lj}^{(2)} \delta_l^{(2)} \quad (3.15)$$

The partial derivative for the hidden layer is computed similarly as for the output layer:

$$\frac{\partial E}{\partial w_{ji}^{(1)}} = x_i \delta_j^{(1)} \quad (3.16)$$

The back-propagated error can be computed in this same way for any number of hidden layers.

**Fourth step: Weights updates**

After computing all partial derivatives the weights are updated as follows:

$$\Delta w_{ji}^{(2)} = \gamma o_i^{(1)} \delta_j^{(2)} , \text{ for } i = 1, ..., k; \ j = 1, ..., m$$

$$\Delta w_{ji}^{(1)} = \gamma x_i \delta_j^{(1)} , \text{ for } i = 1, ..., n; \ j = 1, ..., k$$
It is important to update the weights only after the back propagated errors have been computed for all units in the network. Otherwise, the corrections become intertwined with the back propagation of the errors and the computed corrections would not correspond any more to the negative gradient direction (Rojas, 1996).

During a learning task, the search process is prone to wide oscillations in cases when the minimum of the error function lies in a local narrow "valley". In order to prevent the system from converging to a local minimum, a momentum parameter is used. By introducing the momentum term, the search of the minimum of the error function is oriented toward the centre of the valley (see Figure 3.7).

At each step, the weights are updated by computing the current gradient and the previous weighted correction direction. The insertion of the momentum parameter should provide the search process a kind of inertia and should avoid the oscillations in narrow valleys of the error function. The expression for the \(i\)-th correction for weight \(w_k\) will be the following:

\[
\Delta w_k(i) = -\gamma \frac{\partial E}{\partial w_k} + \alpha \Delta w_k(i - 1) \tag{3.17}
\]
where $\gamma$ and $\alpha$ are the learning rate and the momentum rate respectively. Figure 3.7 shows the difference in the search process of the error function with and without the momentum parameter for a network with just two weights. The introduction of the momentum rate allows the attenuation of oscillations in the learning process.
Chapter 4

Spiking Neuron Models

According to their computational units, neural networks can be classified in three different generations (Maass, 1997a). While the first generation of artificial neuron networks consists of the simple McCulloch-Pitts threshold neurons, the second generation consists of the more powerful neurons which use continuous activation functions. The second generations of neural networks (described in the previous chapter) are called rate neurons, because of the way the output signal correlates with the firing rate.

However, experimental evidence suggests that neural systems use the exact time of single action potentials to encode information (Johansson and Birznieks, 2004; Thorpe and Imbert, 1989). In Thorpe and Imbert (1989) it is argued that because of the speed of processing visual information and the anatomical structure of the visual system, processing has to be done on the basis of single spikes. In Johansson and Birznieks (2004) it is shown that the relative timing of the first spike contains important information about tactile stimuli. Further evidence suggests that the precise temporal firing pattern of groups of neurons conveys relevant sensory information (deCharms and Merzenich, 1996; Neuenschwander and Singer, 1996; Wehr and Laurent, 1996).

These findings have led to a new way of simulating neural networks based on temporal encoding with single spikes (Maass, 1997a). Thus, the third generation
of neural networks is based on neurons that use individual spikes. Investigations of the computational power of spiking neurons have illustrated that realistic mathematical models of neurons can arbitrarily approximate any continuous function, and furthermore, it has been demonstrated that networks of spiking neurons are computationally more powerful than sigmoidal neurons (Maass, 1997b). Because of the nature of spiking neuron communication, these are also suited for VLSI implementation with significant speed advantages (Elias and Northmore, 2002).

Although many mathematical models of spiking neurons exist, these are only simplified models that do not provide a complete description of the complex biological nerve cell. In this chapter the most important and most frequently used models of spiking neurons will be reviewed.

Neural activity may be described at different levels of abstraction (Gerstner, 1999). On a microscopic level, action potentials are the result of ion currents that pass through ion channels in the cell membrane. Compartmental models describe the dynamics of these currents in terms of a set of differential equations which account for the ion channels, different types of synapses, and the specific spatial geometry of an individual neuron. In Section 4.1 the Hodgkin-Huxley model is briefly described and analysed.

On a higher level of abstraction, integrate-and-fire models ignore the spatial structure of the neuron and the exact ionic mechanism, considering the neuron as a homogeneous unit which fires if the total excitation reaches a certain threshold. In Section 4.2 the integrate-and-fire and spike response models are reviewed. On an even higher level of abstraction, there are population models which describe the activity and the interaction between whole brain areas where neuron are organised in populations of units with similar properties.

4.1 Hodgkin-Huxley model

In an extensive series of experiments on the giant axon of the squid, Hodgkin and Huxley (1952) measured the currents that passed through the ion channels and
4. Spiking Neuron Models

determined one of the most important models in computational neuroscience.

Ion channels represent the simplest mechanism by which ions can enter or exit the nerve cell (Kandel et al., 2000). All ion channels share two important properties: firstly, they are selective to the species of ions that may pass through the channel, and secondly, they are gated, meaning that the channel may be switched open or shut. When open, selected ion species may cross the membrane at very high rates; when closed no ions may traverse the channel. One of the mechanisms through which a channel gate can be controlled is by responding to a specific change in voltage across the membrane in the vicinity of the gate (Beatty, 2001). These are called voltage-gated ion channels.

The Hodgkin-Huxley model can be understood through Figure 4.1. The interior of the cell is separated from the extracellular liquid by a semipermeable cell membrane which acts as a capacitor. If an input current $I(t)$ is injected into the cell, it may add further charge on the capacitor, or leak through the channels in the cell membrane.

![Figure 4.1: Schematic diagram for the Hodgkin-Huxley model. Reproduced after Gerstner and Kistler (2002), Figure 2.2.](image)

Applying the conservation of electric charge on the membrane, the current $I(t)$ may be split into a capacitive current $I_C$ which charges the capacitor $C$ and the components $I_k$ which pass through all ion channels:

$$I(t) = I_C(t) + \sum_k I_k(t)$$ (4.1)
4. Spiking Neuron Models

From the definition of the capacitor $C = Q/u$ and the charging current $I_C = Cdu/dt$, where $Q$ is the charge and $u$ is the voltage across the capacitor, the following equation arise:

$$\frac{du}{dt} = I(t) - \sum_k I_k(t)$$  \hspace{1cm} (4.2)

In the nerve cell, $u$ represents the voltage across the membrane and $\sum_k I_k(t)$ is the sum of the ionic currents which pass through the cell membrane. Hodgkin and Huxley (1952) concluded that the squid axon carries three major ion currents: voltage-gated persistent potassium current $I_K$ with four activation gates; voltage-gated transient sodium current $I_{Na}$ with three activation gates and one inactivation gate; there is also a leakage channel with resistance $R$, carried mostly by chloride ions. The channels may be characterized by their resistance $R$ or, equivalently, by their conductance $g = 1/R$. The leakage channel is described by a voltage-independent conductance $g_L = 1/R$, while the conductance of the other ion channels is voltage and time dependent (Gerstner and Kistler, 2002).

The Hodgkin-Huxley model is completely described by the following equations: the first one describes the conservation of electric currents, and the other three differential equations describe the dynamics of sodium and potassium channels.

$$\frac{du}{dt} = I(t) - g_Kn^4(u - E_K) - g_{Na}m^3h(u - E_{Na}) - g_L(u - E_L)$$  \hspace{1cm} (4.3)

$$\dot{n} = \alpha_n(u)(1 - n) - \beta_n(u)n$$  \hspace{1cm} (4.4)

$$\dot{m} = \alpha_m(u)(1 - m) - \beta_m(u)m$$  \hspace{1cm} (4.5)

$$\dot{h} = \alpha_h(u)(1 - h) - \beta_h(u)h$$  \hspace{1cm} (4.6)

where $m$, $n$, $h$ are gating variables that describe the probability that a channel is open, with $\dot{m} = dm/dt$ and so on. The various functions $\alpha$ and $\beta$ describe the transition rates between open and closed states of the channels and they have been determined empirically by Hodgkin and Huxley to fit the data of the giant axon of the squid. The values of the parameters are summarised below, as
reported by Hodgkin and Huxley (1952). These values correspond to a membrane potential shifted by approximately 65 mV, so that the resting potential is zero.

\[ \alpha_n(u) = 0.01 \frac{10 - u}{\exp \frac{10 - u}{10} - 1} \]  
\[ \beta_n(u) = 0.125 \exp \frac{-u}{80} \]  
\[ \alpha_m(u) = 0.01 \frac{25 - u}{\exp \frac{25 - u}{10} - 1} \]  
\[ \beta_m(u) = 4 \exp \frac{-u}{18} \]  
\[ \alpha_h(u) = 0.07 \exp \frac{-u}{20} \]  
\[ \beta_h(u) = \frac{1}{\exp \frac{30 - u}{10} + 1} \]  

The parameters \( E_{Na} \), \( E_K \), and \( E_L \) are the equilibrium potentials, when the difference in concentration of ions across the membrane and the electric potential gradient exert equal and opposite forces that counterbalance each other, and the net cross-membrane current is zero. The membrane equilibrium potential of a cell membrane permeable to one species of ion is given by the Nernst equation:

\[ \Delta u = \frac{kT}{q} \ln \frac{n_2}{n_1} \]

where \( k \) is the Boltzmann constant, \( T \) is the temperature, \( q \) is the charge, and \( n_2 \) is the ion concentration inside the neuron, and \( n_1 \) is the ion concentration of the surround (Gerstner and Kistler, 2002). Equilibrium potentials and conductances have also been determined empirically:

\[ E_K = -12 \text{ mV}, \ E_{Na} = 120 \text{ mV}, \ E_L = 10.6 \text{ mV} \]

Typical values of maximal conductances for a membrane capacitance \( C = 1 \mu\text{F/cm}^2 \) are:

\[ \bar{g}_K = 36 \text{ mS/cm}^2, \ \bar{g}_{Na} = 120 \text{ mS/cm}^2, \ g_l = 0.3 \text{ mS/cm}^2 \]

All neuron models based on the Hodgkin-Huxley equations are called conductance based models. These models are important in computational neuroscience.
because all their variables and parameters have a well-defined biophysical meaning and can be measured experimentally (Izhikevich, 2007). The drawback of conductance based models is that the measurement procedures of the parameters may not be accurate, as these are usually measured in different neurons, averaged and fine-tuned. As a consequence, this model does not always behave as one sees in experiments (Izhikevich, 2007).

4.2 Simple models

Although detailed conductance based models can reproduce electrophysiological measurements to a high degree of accuracy, their intrinsic complexity makes these models difficult to analyse mathematically. Simple phenomenological spiking neuron models are preferred when reproductions of neurocomputational features of neurons are needed. In this section some of most popular models for studying neural coding, memory, and network dynamics are discussed.

4.2.1 The leaky Integrate-and-fire model

The leaky integrate-and-fire neuron is probably the best-known example of a formal spiking neuron model, being an idealisation of a neuron with a leakage current and a number of voltage-gated currents that are deactivated at rest (Gerstner and Kistler, 2002).

\[ \text{Figure 4.2: Schematic diagram of the integrate-and-fire model.} \]
4. Spiking Neuron Models

The basic circuit of an integrate-and-fire model consists of a capacitor $C$ in parallel with a resistor $R$ driven by a current $I(t)$ (see Figure 4.2). The current that charges the $RC$ circuit can be split into its two components, the resistive current and the capacitive current:

$$I(t) = \frac{u(t)}{R} + C \frac{du}{dt}$$

(4.16)

where $u$ is the membrane potential. Introducing the time constant $\tau_m = RC$ on the leaky integrator, results the standard form of the leaky integrator:

$$\tau_m \frac{du}{dt} = -u(t) + RI(t)$$

(4.17)

In integrate-and-fire models action potentials are characterized by the firing time $t^{(f)}$ defined by the threshold condition:

$$t^{(f)} : u(t^{(f)}) = \vartheta$$

(4.18)

When the membrane potential reaches the threshold $\vartheta$, a spike is fired and immediately after that, the potential is reset to a new value $u_r < \vartheta$, giving the expression of the reset:

$$\lim_{t \to t^{(f)}, t > t^{(f)}} u(t) = u_r$$

(4.19)

For $t > t^{(f)}$, the dynamics of the neuron is given by the leaky integration equation (4.19).

The basic integrate-and-fire model is defined by the leaky integration (4.17) and the reset equations (4.19) (Stein, 1967). The generalised version of the model also incorporates an absolute refractory period $\Delta_{abs}$, during which the neuron cannot fire before the membrane potential is reset to the value $u_r$.

The integrate-and-fire model is useful for analytical studies due to its neuro-computational properties. Firstly, the spikes are considered to be identical in size and shape, because the shape of the action potential is not simulated. Secondly, the threshold is well defined, meaning that a spike is fired as soon as the potential reaches the threshold.
There are a number of variations of the integrate-and-fire neuron, such as the nonlinear integrate-and-fire model in which the parameters are dependent on the voltage. A specific instance of the nonlinear integrate-and-fire model is the quadratic model which can capture the spike generation of real neurons (Izhikevich, 2007).

4.2.2 Spike Response Model (SRM)

Another generalisation of the leaky integrate-and-fire model is the Spike Response Model (SRM), where the parameters are dependent on the time since the last output spike (Gerstner, 2001; Gerstner and Kistler, 2002). Instead of defining the model in terms of differential equations, SRM characterises the neuron by a single variable, the membrane potential $u(t)$, at time $t$ as an integral over the past.

The emission of the action potential can be described by a threshold process as follows. An action potential will be fired if the membrane potential $u(t)$ reaches a formal threshold $\theta$ at a time $t^{(f)}$:

$$u(t^{(f)}) = \theta \text{ and } \frac{du(t^{(f)})}{dt} > 0$$ \hspace{1cm} (4.20)

The last firing time of a neuron is defined as $\hat{t} := \max \{t^{(f)} | t^{(f)} < t\}$. If no input is applied at the time $t > t^{(f)}$, the potential trajectory will have a pulse-like excursion before it returns to the resting potential:

$$u(t) = \eta(t - \hat{t}) + u_{\text{rest}}$$ \hspace{1cm} (4.21)

where $u_{\text{rest}}$ is the resting potential and $\eta$ is the standard shape of the pulse with $\eta(t - \hat{t}) \to 0$ for $(t - \hat{t}) \to \infty$. The kernel $\eta$ includes the form of the action potential as well as the after-hyperpolarisation potential, if needed.

If a small input current $I$ is applied at $t' > \hat{t}$ the membrane potential will have another trajectory. If the input current is small enough, the potential can
be described by a linear impulse kernel $\tilde{\epsilon}$ as follows:

$$u(t) = \eta(t - \tilde{t}) + \int_{0}^{\infty} \tilde{\epsilon}(t - \tilde{t}, s) I_{ext}(t - s)ds + u_{rest} \quad (4.22)$$

The Spike Response Model is described by the above equation (Gerstner, 2001).

In the case of a single neuron $i$ receiving input from a set of presynaptic neurons $j \in \Gamma_i$, the state of the neuron is described by the variable $u_i(t)$. In this case, the effects of the incoming spikes are summarised and if $u_i$ reaches the threshold $\vartheta$ an action potential is triggered. Considering the time of the last action potential $\tilde{t}_i$ of neuron $i$, the temporal evolution of the potential is given by:

$$u_i(t) = \eta(t - \tilde{t}_i) + \sum_{j \in \Gamma_i} w_{ij} \sum_{t_j^{(f)} \in \Gamma_j} \epsilon_{ij}(t - \tilde{t}_i, t - t_j^{(f)})$$

$$+ \int_{0}^{\infty} \tilde{\epsilon}(t - \tilde{t}_i, s) I_{ext}(t - s)ds \quad (4.23)$$

where $t_j^{(f)}$ is the time of the action potential of presynaptic neuron $j$ and $w_{ij}$ is the synaptic efficacy. The last term represents an external driving current $I_{ext}$. The sum runs over all incoming spikes where $\Gamma^j$ is the set of all firing times $t_j^{(f)} < t$ of neuron $j \in \Gamma_i$, where $\Gamma_i = \{ j | j \ \text{presynaptic to} \ i \}$.

The above equation together with the threshold criterion defines the formal model (Gerstner, 2001).

**4.3 Discussion**

Although the Hodgkin-Huxley model is based on experimental measurements and their parameters are biologically meaningful, these need 1200 floating point operations (Izhikevich, 2004). On the other hand, the much simpler leaky integrate-and-fire only needs 5 floating point operations to simulate. Even though the these simple models do not exhibit all spiking behaviour seen in real neurons, their reduced computational complexity make them ideal for simulations where
only phenomenological properties are required.
Chapter 5

Learning in Spiking Neural Networks

As seen in the previous chapter, the amplitude of the postsynaptic response to an incoming action potential is mainly determined by the strength of the synapse. Electrophysiological experiments show that appropriate stimulation paradigms can systematically induce changes of the amplitude of the postsynaptic response. Persistent changes such as LTP and LTD (see Section 2.4) are thought to be the neuronal correlate of "learning" and "memory" (Gerstner and Kistler, 2002). In this chapter the most important form of Hebbian learning for spiking neurons is presented as well as the existing supervised learning methods for spiking neural networks.

5.1 Spike-Time Dependent Plasticity (STDP)

One of the simplest procedures for adjusting the weights is given by synaptic changes driven by correlated activity of pre- and postsynaptic neurons. This class of learning rule is based on Hebb's principle (Hebb, 1949) and is therefore often called "Hebbian learning".

Hebb's postulate (Hebb, 1949) says that if a presynaptic neuron i takes part in
5. Learning in Spiking Neural Networks

exciting a postsynaptic neuron \( j \), some process takes place in one or both cells such that the synaptic efficacy between the two cells is increased. Over time, Hebbian learning generally refers to the modifications in the synaptic transmission efficacy that are driven by correlations in the firing activity of pre- and postsynaptic neurons (Gerstner and Kistler, 2002).

As Hebb’s postulate requires both pre- and postsynaptic neurons to be active in order to induce a change in the synaptic efficacy, the timing conditions for synaptic changes have been measured in order to determine the temporal requirements for weight changes (Gerstner and Kistler, 2002). A pairing experiment with cultured hippocampal neurons (Bi and Poo, 1998) showed that the resulting change in the synaptic efficacy is dependent on the spike time differences \( t_{j}^{(f)} - t_{i}^{(f)} \), where \( t_{j}^{(f)} \) is the postsynaptic spike time and \( t_{i}^{(f)} \) is the presynaptic spike time. The synapse efficacy is increased if the presynaptic spike occurs shortly before the postsynaptic neuron fires, but the synapse efficacy is decreased if the sequence of spikes is reversed. This process is called spike-time dependent synaptic plasticity (STDP).

Based on the experiments by Bi and Poo (1998), Kistler and van Hemmen (2000) developed a phenomenological model for STDP. Weight changes are considered instantaneous and apart from an activity-independent weight decay all changes are triggered by pre- and postsynaptic action potentials. The input and output signals are spike trains described by a sequence of firing times \( t^{j} \) as a sum of Dirac \( \delta \) functions (Gerstner and Kistler, 2002):

\[
S(t) = \sum_{j} \delta(t - t^{j})
\]  

The total weight change is given by the effect of presynaptic spikes as well as postsynaptic spikes:

\[
\frac{d}{dt} w_{ij}(t) = a_{0} + S_{j}(t) \left[ a_{pre} + \int_{0}^{\infty} a_{pre,post}(s) S_{i}(t - s) ds \right] \\
+ S_{i}(t) \left[ a_{post} + \int_{0}^{\infty} a_{post,pre}(s) S_{j}(t - s) ds \right] 
\]  

(5.2)
The integration variable $s = t_j^{(f)} - t_i^{(f)}$ represents the delay between the pre- and postsynaptic spikes. The non-Hebbian terms $a^{\text{pre}}$ and $a^{\text{post}}$ may depend on the actual value of the weight. The integral kernels $a^{\text{pre,post}}$ and $a^{\text{post,pre}}$ define the learning window and these also may depend on the actual value of the weight (Kistler and van Hemmen, 2000):

$$W(s) = \begin{cases} a^{\text{post,pre}}(-s) & \text{if } s < 0 \\ a^{\text{pre,post}}(s) & \text{if } s > 0 \end{cases} \quad (5.3)$$

In the context of networks of neurons stimulated by input with certain statistical properties, synaptic plasticity that generates changes in the connectivity between neurons that reflect the statistical structure of the input is referred to as unsupervised learning.

### 5.2 Supervised learning methods

Apart from unsupervised mechanisms of learning, learning from instructions or demonstrations is also fundamental for developing new skills and acquiring new knowledge (Ponulak and Kasinski, 2010). Such a supervised method of learning should enable a spiking neural network to reproduce arbitrary patterns of spikes in response to a given input signal. In this section some of the representative supervised learning algorithms for spiking neurons are reviewed.

#### 5.2.1 The tempotron

Gütig and Sompolinsky (2006) introduced a simple supervised learning rule based on the gradient-descent approach, called the tempotron. Spiking neurons are trained to discriminate between spatiotemporal sequences of spike patterns. The tempotron is used to train a leaky integrate-and-fire neuron described by a generalised form of the model. The neuron is described by the expression of the
sub-threshold membrane potential as a weighted sum of incoming signals:

\[ u(t) = \sum_i w_i \sum_{t_i} K(t - t_i) + u_{rest} \]  

(5.4)

where \( t_i \) are the spike times of input neuron \( i \) and \( K(t - t_i) \) is the normalised postsynaptic potential (PSP) for each incoming spike:

\[ K(t - t_i) = u_0 \left( \exp \left(-\frac{t - t_i}{\tau_s}\right) - \exp \left(-\frac{t - t_i}{\tau_s}\right) \right) \]  

(5.5)

The parameters \( \tau \) and \( \tau_s \) represent the decay time constants. The factor \( u_0 \) normalises the PSP kernel to 1. If the membrane potential reaches the threshold, an action potential is fired and the potential is reset to the resting potential \( u_{rest} \).

The learning rule is used to classify input patterns into two classes. The output neuron learns to discriminate the incoming spike trains by responding with at least one action potential for one class of patterns, and by remaining quiescent for the other. The weights are modified whenever an input pattern is misclassified:

\[ \Delta w_i = \gamma \sum_{t_i < t_{\text{max}}} K(t_{\text{max}} - t_i) \]  

(5.6)

where \( t_{\text{max}} \) is the time at which the postsynaptic membrane potential \( u(t) \) reaches its maximal value. The learning parameter \( \gamma > 0 \) denote the maximum weight modification for each input spike. The weights are potentiated by an amount given by equation (5.6) if the neuron fails to elicit a spike when an action potential is required, or depressed by the same amount if a spike if fired when the output neuron is required to remain quiescent.

Although the learning rule has been successfully used to classify spike timing patterns, the neurons do not learn to respond with precise spike-timing patterns. Because the output neurons do not learn to respond with spike timing patterns, the learning rule cannot be extended to networks with multiple layers.
5.2.2 ReSuMe - Remote Supervision

Another method based on Hebbian learning is ReSuMe - Remote Supervision (Ponulak and Kasiński, 2010). The goal of the algorithm is to train a neural network to produce a desired spike train in response to a given input pattern of spikes. The authors argue that the method possesses the interesting properties of the supervised Hebbian learning, while avoiding its drawbacks (Kasiński and Ponulak, 2006).

Ponulak and Kasiński (2010) considered the Widrow-Hoff \textit{delta} rule (see Section 3.1.3) as a starting point for the derivation of the learning rule. The input and output signals are spike trains described by a sequence of firing times as a sum of δ functions (see equation (5.1)). The \textit{delta} rule is formally rewritten using the spike train signals in place of the activation functions:

\[
\frac{d}{dt} w(t) = S_i(t)S_d(t) - S_i(t)S_o(t)
\]

(5.7)

where \(S_i(t)\), \(S_o(t)\), and \(S_d(t)\) are the input, output and target spike train respectively. The two products, \(S_d(t)S_i(t)\) and \(S_o(t)S_i(t)\), are interpreted as STDP and anti-STDP processes:

\[
S_i(t)S_d(t) \rightarrow S_i(t) \left[ a + \int_{0}^{\infty} a_{pre}(s)S_d(t-s)ds \right] \\
+ S_d(t) \left[ a + \int_{0}^{\infty} a_{post}(s)S_i(t-s)ds \right]
\]

(5.8)

\[
S_i(t)S_o(t) \rightarrow S_i(t) \left[ a + \int_{0}^{\infty} a_{pre}(s)S_o(t-s)ds \right] \\
+ S_o(t) \left[ a + \int_{0}^{\infty} a_{post}(s)S_i(t-s)ds \right]
\]

(5.9)
where $a > 0$ is a non-Hebbian term. The final formula for weight update becomes:

$$
\frac{d}{dt} w(t) = S_i(t) \int_0^\infty a^{pre}(s) \left[ S_d(t - s) - S_o(t - s)ds \right] + (S_d(t) - S_o(t)) \left[ a + \int_0^\infty a^{post}(s) S_i(t - s)ds \right]
$$

(5.10)

In case of excitatory synapses, the term $a$ is positive, and in case of inhibitory synapses the term $a$ is negative (Kasiński and Ponulak, 2006). The kernels $a^{pre}$ and $a^{post}$ define the learning window $W(s)$, which determines the amplitude of the change (Gerstner and Kistler, 2002):

$$
W(s) = \begin{cases} 
    a^{pre}(-s) = -A_- \exp\left(\frac{s}{\tau_-}\right), & \text{if } s \leq 0 \\
    a^{post}(s) = +A_+ \exp\left(\frac{s}{\tau_+}\right), & \text{if } s > 0
\end{cases}
$$

(5.11)

where $A_+, A_- > 0$ are the amplitudes and $\tau_+, \tau_- > 0$ are the time constants of the learning window.

As the weight modifications are based only on the input and output spike trains and do not make any explicit assumptions about the neural or synaptic dependencies, the algorithm can be applied to various neuron models. The learning method was successfully tested with leaky integrate-and-fire and Hodgkin-Huxley neuron models and it is argued that it can be used in networks with combined different neuron models (Kasiński and Ponulak, 2006). However, the algorithm can only be applied to a single layer of neurons or used to train readouts for reservoir networks.

ReSuMe has also been used to train neural networks with a hidden layer, where weights of downstream neurons are subject to multiplicative scaling (Grüning and Sporea, 2012). The simulations show that networks with one hidden layer can perform non-linear logical operations, while networks without hidden layers cannot. Other studies also showed that the output layer in feedforward networks can be trained with ReSuMe, where the hidden layer acts as a frequency filter (Glackin et al., 2011; Wade et al., 2010). However, input and target outputs consisted of fixed-rate spike trains.
5. Learning in Spiking Neural Networks

5.2.3 SpikeProp - Gradient based learning

In feed-forward neural networks learning is usually performed by gradient descent techniques, like the back-propagation described in the previous chapter (see Section 3.2.2). In spiking neural networks, such methods are difficult to apply due to the discontinuity in time of the spiking neurons. Bohte et al. (2002) proposed such a method, SpikeProp, which is similar to the traditional back-propagation. The discontinuity of the membrane potential is avoided by considering that each neuron can fire only one spike and by approximating the threshold function.

The learning method has been derived for neurons modelled by the Spike Response Model (Gerstner, 2001). The network architecture consists of a feed-forward network of spiking neurons, with each connection between two neurons having a number of $m$ synaptic terminals, or sub-connections, with different delays, $d^k$, between the firing time of the presynaptic neuron and the time the postsynaptic potential starts to rise or to drop if the neuron is excitatory, or respectively inhibitory (neurons generating only negative postsynaptic potentials). The membrane potential of one neuron $j$ is thus described as follows:

$$x_j(t) = \sum_{i \in \Gamma_j} \sum_k w^k_{ji} y^k_i(t)$$  \hspace{1cm} (5.12) $$

where $\Gamma_j$ is the set of all presynaptic neurons of neuron $j$. The term $w^k_{ji}$ represents the weight of the synaptic terminal $k$, having the delay $d^k$, between the neurons $i$ and $j$. $y^k_i(t)$ is the unweighted contribution of a single sub-connection:

$$y^k_i(t) = \epsilon(t - t_i - d^k)$$  \hspace{1cm} (5.13) $$

where $t_i$ is the firing time of presynaptic neuron $i \in \Gamma_j$. The spike-response function $\epsilon(t)$ describes a standard postsynaptic potential and has the form:

$$\epsilon(t) = \begin{cases} \frac{t}{\tau} \exp \left(1 - \frac{t}{\tau} \right), & \text{if } t > 0 \\ 0, & \text{if } t \leq 0 \end{cases}$$  \hspace{1cm} (5.14) $$

with $\tau$ modelling the membrane potential decay time constant.
5. Learning in Spiking Neural Networks

The learning method consists of explicitly evaluating the error gradient with respect to the weights of each synaptic terminal, where the error function is defined as:

$$E = \frac{1}{2} \sum_j (t^a_j - t^d_j)^2$$  \hspace{1cm} (5.15)

Thus, the weights are modified in the direction of the gradient descent:

$$\Delta w^k_{ji} = -\gamma \frac{\partial E}{\partial w^k_{ji}}$$  \hspace{1cm} (5.16)

where \(\gamma\) is the learning rate. The error gradient is further expanded using the chain rule:

$$\frac{\partial E}{\partial w^k_{ji}} = \frac{\partial E}{\partial t^i_j} \frac{\partial t^i_j}{\partial x^i_j(t)} \frac{\partial x^i_j(t)}{\partial w^k_{ji}}$$

where \(t^i_j\) is the actual firing time of the postsynaptic neuron \(j\). In order to assure the continuity of the spiking neuron, it is assumed that for a small enough region around \(t = t^i_j\), the function \(x^i_j(t)\) can be approximated by a linear function of \(t\).

Bohte et al. (2002) also presented a series of results that demonstrate the learning capabilities of SpikeProp. The Spike prop algorithm is tested with the classic linearly non-separable XOR problem. The XOR function is encoded in
5. Learning in Spiking Neural Networks

spike-time patterns by associating a 0 with a "late" firing time, and a 1 with an "early" firing time. Although not a lot of details were given with respect to the initial training parameters, Bohte et al. (2002) affirmed that a network with five hidden neurons, one of which being inhibitory, was able to learn the XOR problem within 250 cycles.

The algorithm was also tested with other benchmark problems, such as the Iris dataset, the Wisconsin breast-cancer dataset, and the Statlog Landsat dataset. The SpikeProp algorithm was compared against a Matlab implementation of a sigmoidal neural network trained with the default methods Levenberg-Marquardt and back-propagation. The number of neurons was chosen the same for all three types of learning methods. Although in all three benchmark problems SpikeProp required less cycles for the algorithm to converge than the classical learning methods, the time required for a single learning iteration in SpikeProp is considerably longer, since there are 16 times more weights to be updated.

**Improvements of SpikeProp convergence time**

Modifications of SpikeProp have been proposed in order to speed up the convergence (McKennon et al., 2006; Xin and Embrechts, 2001). While Xin and Embrechts (2001) introduced a momentum term in the weight update equation, McKennon et al. (2006) developed the QuickProp and RProp versions of SpikeProp for faster convergence times tested on the Iris data set. Thorough investigations of the shape of error in terms of the parameters of the network (Fujita et al., 2008; Takase et al., 2009) show that the learning curve fluctuates regardless of the progress of learning. Because of these fluctuations, Fujita et al. (2008) argue that acceleration methods would fail, as inaccurate local information would lead to incorrect estimation of the global shape of error.

Another version of SpikeProp (Schrauwen and Van Campenhout, 2004) included also the adjustment of the synaptic delays, time constants and neuron thresholds. While this method results in smaller network topologies, as it requires only two synaptic terminals instead of 16, and the algorithm converged faster, the adaptation of these parameters introduce unnecessary computations
as Bohle et al. (2002) demonstrated that the time constant can be optimally chosen in terms of the encoding of the spike-time patterns. Similar demonstrations exist for the selection of the threshold in terms of the initial weight values (Moore, 2002). Moreover, the adjustment of the synaptic weights, synaptic delays, time constants, and thresholds require different learning rates which introduces further problems in choosing and combining their optimal values – Moore (2002) demonstrated that a careful choice of the learning rate in terms of the initial values of weights is essential for the convergence of the algorithm.

Learning multiple spikes

An important development of the SpikeProp algorithm was introduced by Booij and Nguyen (2005) who proposed a learning rule for neurons firing multiple spikes.

The neurons are modelled according to the Spike Response Model (Gerstner, 2001) as in spiking neural networks trained with SpikeProp. As multiple spikes are allowed, the kernel \( \eta \) is introduced to describe the form of the potential after a spike is emitted:

\[
u_i(t) = \sum_{t'_i \in F_i} \eta(t - t'_i) + \sum_{j \in F_i} \sum_{k} w_{ij}^k \epsilon(t - t'_j - d^k) \quad (5.20)
\]

where \( F_i = \{ t'_i, 1 < f \leq n \} \) is the spike train of neuron \( i \) and \( n \) denotes the number of spikes chronologically ordered: \( 1 \leq f < g \leq n \rightarrow t'_i < t'_j \). Booij and Nguyen (2005) used as the spike response function \( \epsilon \) the difference between two exponential decays:

\[
\epsilon(t) = \left[ \exp\left( -\frac{t}{\tau_m} \right) - \exp\left( -\frac{t}{\tau_s} \right) \right] H(t) \quad (5.21)
\]

where \( H(t) \) is the Heavyside step function:

\[
H(t) = \begin{cases} 
0, & \text{for } t \leq 0 \\
1, & \text{for } t > 0
\end{cases} \quad (5.22)
\]
5. Learning in Spiking Neural Networks

The time constants $\tau_m$ and $\tau_s$ determine the rise and decay of the function. The kernel $\eta$ models the refractoriness of the neuron after it emitted a spike and it is expressed by an exponential decay:

$$\eta(t) = -\vartheta \exp \left( -\frac{t}{\tau_r} \right) H(t) \quad (5.23)$$

where $\tau_r$ is the time constant, and $\vartheta$ is the membrane threshold. In the learning process, the derivative of these functions for $t = 0$, are undetermined and hence are approximated with zero.

In the learning process, although the algorithm is taking into account neurons that can fire multiple spikes, the network error is defined in terms of the first spike of the output neurons, later spikes of these neurons being ignored. The weight modifications are now calculated with respect to all fired spikes in the input and hidden layers.

While the second factor is determined in the same manner as in the SpikeProp algorithm, the derivative of the network error with respect to a spike is calculated for every spike of non-output neuron $i$ as follows:

$$\Delta w_{ij}^k = -\gamma \sum_{t'_i \in F_i} \frac{\partial E}{\partial t_i^j} \frac{\partial f_i}{\partial w_{ij}^k} \quad (5.24)$$

The derivative of the postsynaptic spike with respect to the presynaptic spike is calculated as:

$$\frac{\partial t_j^g}{\partial t_i^j} = \frac{\partial u_i(t_j^g)}{\partial t_i^j} \frac{-1}{\partial u_i(t_j^g)} \quad (5.25)$$

where the first factor of the formula is calculated recursively:

$$\frac{\partial u_i(t_j^g)}{\partial t_i^j} = -\sum_{t_j^g \in F_j} \eta'(t_j^g - t_i^j) \frac{\partial t_i^j}{\partial t_i^j} - \sum_k w_{ij}^k \epsilon(t_j^g - t_i^j - d^k) \quad (5.26)$$
5. Learning in Spiking Neural Networks

In order to overcome some of the problems introduced in SpikeProp, Booij and Nguyen (2005) also introduce two additional conditions. The gradient of the potential is lower bounded so that when the potential gradient is very small, the derivative of the error would not produce a very large weight change. Furthermore, to avoid the case when the weights are so small that the output neurons would not spike, and the error would become zero, the learning is started with sufficiently high weights that the neurons will initially fire for all input patterns.

The learning algorithm for spiking neural networks with multiple spikes was trained with two sets of Poisson spike trains patterns (Heeger, 2000). The output of the network consisted of a single neuron designated to distinguish the two classes of spiking patterns. The algorithm converged on average in 17 learning cycles with an accuracy of 89%. However, the algorithm was not tested with more complex inputs consisting of more than two classes of patterns.

Another implementation of the SpikeProp algorithm for neurons emitting multiple spikes was developed by Ghosh-Dastidar and Adeli (2009). The learning algorithm is determined in a similar manner as Booij and Nguyen's (2005) approach for neurons emitting multiple spikes in the input and hidden layer, the output neurons being restricted to one spike. However, in this implementation, the refractoriness kernel $\eta$ is calculated in terms of only the most recent spike, instead of being summed over all spikes.

5.2.4 Other supervised learning methods

Other supervised learning algorithms are analysed and compared in Kasiński and Ponulak (2006). The learning methods include supervised rules based on Hebb's postulate, statistical criteria and linear algebra. These algorithms are revised in this section.

Using leaky integrate-and-fire neurons Ruf and Schmitt (1997) developed a supervised learning rule based on Hebb's principle. The proposed algorithm was defined for a single excitatory synapse as follows: Considering two neurons, $u$ and $v$, connected by an excitatory synapse, the learning rule is based on three
spikes: two presynaptic, at times $t_0$ and $t_u$, and a postsynaptic spike at time $t_v$. The time interval between the two presynaptic spikes, $t_0 - t_u$, is regarded as the input signal for postsynaptic neuron $v$. The additional presynaptic spike, $t_0$, has the purpose of pointing the target firing time for the postsynaptic neuron. The proposed learning rule minimises the difference between the postsynaptic firing time, $t_v$, and the presynaptic reference firing time, $t_0$:

$$\Delta w = \gamma(t_v - t_0)$$

(5.28)

where $\gamma > 0$ is the learning rate. The time difference $t_v - t_0$ is considered the error that will be reduced through changes of the weights.

The Hebbian based learning rule can be successfully used to train single weights, as it was demonstrated that under certain conditions, $t_v$ converges to $t_0$ (Ruf and Schmitt, 1997). However, neurons usually receive input from several presynaptic neurons $u_i$, and the only method that the learning rule can be applied is if the corresponding weights were learnt in parallel. The learning rule for the parallel version of the algorithm is:

$$\Delta w_i = \gamma(t_0 - t_{u_i})$$

(5.29)

where $t_{u_i}$ are the firing times of neuron $u_i$. The weights $w_i$ are normalised after each application of the rule, such that $||w_i|| = 1$, where $||w_i||$ is the Euclidean norm. Again, it was demonstrated that using the above learning rule, the algorithm converges. However, this method can be used to learn a target weight vector for any presynaptic neuron, rather than achieving a desired timing of the postsynaptic neuron. No experimental results confirm these theoretical findings.

Legenstein et al. (2005) proposed a learning rule based on spike-time dependent plasticity for biologically realistic models of spiking neurons. The goal of the algorithm is to map an arbitrary given map of input spike trains to output spike trains where the target spike timing patterns are given by additional input currents provided to the postsynaptic neuron. Thus, the postsynaptic neuron receives a teaching input, which is a depolarising current if the neuron was supposed to fire, or a hyperpolarising current if the neuron fires undesired spikes.
Their modified version of STDP is considered only for excitatory connections and takes the following form:

\[
\Delta w = \begin{cases} 
+W_+(1 - w)^\mu \exp -\frac{s}{\tau_+} & \text{if } s > 0 \\
-W_- w^\mu \exp \frac{s}{\tau_-} & \text{if } s \leq 0 
\end{cases}
\]  

(5.30)

where the constants \( W_+, W_-, \tau_+, \tau_- > 0 \), and \( s \) is the delay between the pre- and postsynaptic spikes, and \( \mu \) is a non-negative exponent.

The leaky integrate-and-fire model was used once again for simulations. Computer simulations have shown that using this learning rule, a neuron can learn a transformation that maps 100 input spike train to one target spike train, with and without noise. Learning multiple input-target mapping was not tested with this algorithm.

Although the results presented in Legenstein et al. (2005) demonstrated that a given target transformation was learnt using the above rule and that the method can converge in the average case for arbitrary uncorrelated input spike trains, the authors showed that convergence cannot be guaranteed in the general case.

Pfister et al. (2003, 2006) proposed a learning method based on statistical criteria with similar results to spike-time dependent plasticity (Bi and Poo, 1998). The aim of the algorithm is to optimise the likelihood of having a postsynaptic spike at the desired time. In order to define the likelihood of a spike train, given a certain input spike train, the authors assume that the firing rate \( \rho(t) \) of a postsynaptic neuron is determined by an increasing function of the membrane potential \( u(t) \):

\[
\rho(t) = g(u(t))
\]  

(5.31)

The likelihood of an output spike train \( S^o(t) \) to be emitted before \( t \) for a given input spike train \( S^i(t) \) is explicitly defined based on the probabilistic approach
5. Learning in Spiking Neural Networks

as follows:

\[ P(S^o(t)|S^i(t)) = \exp\left[ \int_0^T \log(\rho_j(s|S^i(t), S^o(s)))\hat{S}^o(s)
\right.

\[ \left. - \rho(s|S^i(t), S^o(s))ds \right] \]

where \( \hat{S}^o(t) \) is the complete set of postsynaptic spike trains.

The learning algorithm tends to update the weights in the direction of the increased likelihood of getting postsynaptic spikes using a gradient ascent rule:

\[ \Delta w = \gamma \frac{\partial \log P}{\partial w} \]

where \( \gamma \) is the learning rate. As the likelihood is a continuous function, it can be differentiated with respect to the synaptic weight, \( w \). The weights are thus modified in the direction of gradient ascent of the likelihood of the postsynaptic neuron to fire at the desired times.

This learning method also assumes that the postsynaptic neuron receives an additional teaching input, with the purpose of increasing the probability that the postsynaptic neuron will fire closer to the target firing time. Kasiński and Ponulak (2006) compared this mechanism with a probabilistic version of the supervised Hebbian learning described above. In their experiments, Pfister et al. (2006) used postsynaptic spike trains of at most two spikes, for which the output neuron was able to accurately learn the set of times of spikes. However, complex spike trains were not tested, therefore the capabilities of the learning algorithm are not completely identified.

Carnell and Richardson (2005) proposed two simple learning algorithms that aim to approximate a given time series of spikes, when it is formed by a linear combination of a specified collection of time series. The time series is defined as the finite sum of spikes:

\[ \sum_i s(t_i, t) \]

(5.34)
where the spike $s(t_i, t)$ at time $t_i$ is a function of time $t$ defined as follows:

$$s(t_i, t) = \begin{cases} 
1, & \text{if } t = t_i \\
0, & \text{if } t \neq t_i 
\end{cases}$$  \hspace{1cm} (5.35)$$

The weighted time series is defined as:

$$S(t) = \sum_i w_i s(t_i, t)$$  \hspace{1cm} (5.36)$$

where the weights $w_i$ and the times $t_i$ are real numbers. Further, the inner product of two time series is defined as follow:

$$\langle \sum_i w_i s(t_i) \sum_j w_j s(t_j) \rangle = \sum_{i,j} w_i w_j \exp \left( -||t_i - t_j|| \right)$$  \hspace{1cm} (5.37)$$

The operation of projecting a weighted time series $S_1(t)$ onto the direction of another weighted time series $S_2(t)$ is defined as:

$$\text{Proj}_{S_1(t)}(S_2(t)) = \frac{\langle S_1(t) | S_2(t) \rangle}{\langle S_1(t) | S_1(t) \rangle} S_1(t)$$  \hspace{1cm} (5.38)$$

Using a single leaky integrate-and-fire neuron, the methods based on linear algebra aim to approximate a target time series pattern $S^d(t)$, given a input pattern $S^i(t)$ (Carnell and Richardson, 2005). The first method uses a Gram-Schmidt process to find an orthogonal basis for the subspace spanned by a set of the input time series $S^i(t)$. This learning rule is guarantied to find an optimal approximation to any given element in the target time series pattern.

The second solution proposed by Carnell and Richardson (2005) is an iterative algorithm which determines the projection of an error onto the direction of the input time series, where the error is defined as the difference between the target and the actual time series. The tests performed by Carnell and Richardson (2005) showed that this method is able to approximate the target time series.

Feed-forward networks of spiking neurons similar to the one used in Bohte et al. (2002) have also been trained using evolutionary algorithms (Hagras et al.,
5. Learning in Spiking Neural Networks

2004; Jin et al., 2007; Pavlidis et al., 2005). In Pavlidis et al. (2005) a differential evolution method is proposed where weights are evolved through migration and combination of randomly chosen weight vectors. The algorithm starts with a number of sub-populations of weight vectors randomly initialised. During each iteration $g$, in each sub-population $k$, a new weight vector $W^k_{g+1}$ is calculated as follows:

$$W^k_{g+1} = W^k_g + \mu(W^\text{best}_g - W^k_g) + \mu(W^{r_1}_g - W^{r_2}_g)$$  \hspace{1cm} (5.39)

where $W^\text{best}_g$ is the best weight vector found in the previous iteration. $r_1$ and $r_2$ are two random number different from $k$. $\mu$ is the mutation constant which controls the amplitude of the mutation. The algorithm combines the previous weight vector with the best weight vector found so far and two other random vector. However, the elements of the new vector are further recombined with the target weight vector with a certain probability. During the selection process, the resulting vector is accepted only if it yields a smaller error function.

The algorithm has been tested on the XOR problem, the diabetes and Iris datasets using three layer neural networks. Unlike in Bohte et al. (2002), this time the input logical patterns have been encoded with a spike train with 10 spikes or no spike at all, respectively. The output is represented by at least one spike, or no spike at all. Although the network evolves to correctly classify the patterns, in real world problem the target weight vector is not always available to use in the training algorithm. The authors do not give any indication whether the algorithm would still converge without the recombination with the target weight vector.

In Jin et al. (2007), a Pareto-based multi-objective genetic algorithm is used to evolve the connectivity of a feed-forward network, its weights and synaptic delays. These parameters are evolved using uniform crossover and bit-flip mutation. After two generations of populations are combined, these are selected using the crowded tournament method. As such, each individual in the resulting population is assigned a rank and crowding distance and the solution with the best rank is accepted.

The algorithm has been tested on two classification benchmarks, the Wiscon-
sin breast cancer and the diabetes data sets. The patterns have been encoded using single spikes. By evolving the synaptic weights and delays as well as the connectivity of the network, the network is classifying the cancer data with a small enough error with only 5 single terminal connections. As the number of connections increases, the error function continues to slowly decrease to an acceptable level. When using multiple delayed terminals, results are similar but the resulting network evolved less connections. The authors reported that no overfitting was observed in either cases.

Genetic algorithms have also been used for a two layer network of spiking neurons that process ultrasound sensory data for robot control (Hagras et al., 2004). In order to prevent the algorithm reaching a local minimum, the crossover and mutation parameter are varied during the evolution of the synaptic weights. Their experiments showed that using a population of four sets of weights, the genetic algorithm required only 14 iterations (40 minutes of the robot’s time) for the robot to achieve a right edge following behaviour.
Chapter 6

Analysis of SpikeProp

In this chapter, one of the existing supervised learning algorithms, SpikeProp, for feed-forward networks of spiking neurons is analysed. The learning rule derivation is based on the dynamics of the neuron model, SRM (see Section 5.2.3 for details), and as such the learning algorithm is limited to spiking neurons described by SRM.

While Bohte et al. (2002) demonstrated the capabilities of the SpikeProp algorithm the authors give no details on the initial values of the weights. Although some work has been done in this direction (Moore, 2002; Takase et al., 2009) it is still not clear how the network and learning parameters - initial weights, threshold, time constant, learning rate - and the combinations between them affect the ability of the learning method to converge and the number of iterations needed for convergence. These will be analysed in Section 6.1.

Little is also known about the capabilities and limitations of a spiking neural network of this type trained with SpikeProp. Although tested with various benchmark problems by Bohte et al. (2002), no other studies with different benchmark tests or different network structures were performed. In Section 6.2 a different network configuration is considered for the XOR problem. Part of this work has been published in Sporea and Grüning (2011).
6. Analysis of SpikeProp

6.1 SpikeProp convergence

The experiments conducted in Moore (2002) with different values for the initial weights and the learning rate showed that these parameters and the way they are combined are critical for the convergence of the learning process. Since in Bohte et al. (2002) the values for the initial weights and the threshold are not provided, Moore (2002) demonstrated that the threshold and the initial weights must be chosen so that the output neurons would fire for all input patterns. A threshold too high would cause the neurons not to fire, and the network error would be zero, preventing the network to learn. If the threshold is too low, the neurons would fire as soon as one input neuron emitted a spike, and would cause the weights to reach their lower limit, zero. However, the weights were not actually limited to zero in Moore’s experiments and there is no evidence that a too low threshold would prevent the network to learn. An important aspect of the spike-time learning method is the value of the learning rate. Although Bohte et al. (2002) reported that a learning rate of 0.01 was sufficient for the XOR problem, Moore (2002) demonstrated that different scales of values for the weights required different learning rates for an optimal learning time. This finding was also confirmed in Takase et al. (2009) where different learning rates were used with the same initial weights, resulting in different learning times.

6.1.1 Experimental setup

In order to understand how the parameters of the spiking neural network affect the learning process, a series of simulations are conducted for the XOR problem. The structure of the network is identical to the one in (Bohte et al., 2002): three input neurons, one of which designates the reference start time, five hidden neurons, and one output neuron. One of the hidden neurons is inhibitory, while all the other neurons in the network are excitatory. Only positive initial weights are allowed, as Bohte et al. (2002) findings suggested that the algorithm would not converge if negative and positive weights were allowed. However, the weights are allowed to become negative during the learning process.
The input and output patterns are encoded using spike-time patterns as in Bohte et al. (2002). The signals are associated with single spikes as follows: a binary symbol "0" is associated with a late firing (a spike at 6 ms for the input pattern) and a "1" is associated with an early firing (a spike at 0 ms for the input pattern). Table 6.1 shows the input and target spike timing patterns that are presented to the network for the XOR problem. The values represent the times of the spikes for each input and target neuron in ms of simulated time.

Table 6.1: Input and output spike-time patterns. The patterns consist of the timing of single spikes in ms of simulated time for the input and target neurons.

<table>
<thead>
<tr>
<th>Input [ms]</th>
<th>Output [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>16</td>
</tr>
<tr>
<td>0 6 0</td>
<td>10</td>
</tr>
<tr>
<td>6 0 0</td>
<td>10</td>
</tr>
<tr>
<td>6 6 0</td>
<td>16</td>
</tr>
</tbody>
</table>

For the same initial weight values - random weights, within the range of 0.0 and 1.0 uniformly distributed - different thresholds and learning rates were used to test the SpikeProp algorithm. The threshold was varied between 0.2 and 1.1 with a step of 0.1, and the learning rate was varied between 0.001 and 1.5. In total, 80 experiments were conducted. All results of the tests performed on SpikeProp can be found in Appendix A.

The time window for each iteration is set to 50 ms, which proved large enough in order not to affect the learning process. The learning process is stopped when the maximum number of iterations, 5000, is reached or when the network error becomes sufficiently small, 0.5. The time constant in equation 5.14 is set to \( \tau = 0.7 \) as it was suggested to be optimal for the coding interval of \( T = 6 \) (Bohte et al., 2002). During one iteration, all four pattern pairs are presented in a random order. The membrane potential of all neurons in the hidden and output layers is set to the resting potential (set to zero) when presenting a new input pattern. After each presentation of the input pattern to the network, the weight modifications are computed for all layers and then applied. We apply the weight...
6. Analysis of SpikeProp

changes after the backpropagated error is computed for all units in the network. The summed network error is calculated for all patterns and tested against the required minimum value.

6.1.2 Results

For each parameter set, the results are averaged over 50 trials, each with different random initial weights. The charts below summarise the obtained results.

Figure 6.1 shows the percentage of successful learning trials for the XOR problem. A learning trial is considered successful if the network error reaches the minimum error within the maximum number of iterations. The learning fails to converge if the output neuron does not fire for part or all input patterns, as the weight modifications depend on the output firing time. The results confirm that the learning method converges within the maximum number of iterations for certain combinations of initial weights, threshold and learning rate.

The optimal learning rate depends on the threshold as follows: for low thresholds - up to 0.9 - the optimal learning rate reside within the range 0.05 and 0.01; for thresholds with higher values - greater than 0.9 - the convergence rate is higher if the learning rate is around the value of 0.1.

Figure 6.2a shows the average number of iterations needed for convergence for those cases when the learning converges. The results are consistent with other studies (Bohte et al., 2002; Takase et al., 2009) which suggest that the average number of iterations increases as the learning rate decreases. The minimum average of learning iterations was 260 for the threshold 0.4 with the learning rate of 1.0. However the learning algorithm converged only 66% of times.

The average number of iterations does not illustrate how the number of iterations develops during the single trials. Figure 6.3 shows the standard deviation of the number of iterations needed for convergence. Although the average number of iterations for the XOR problem is similar to Bohte et al. (2002) findings, the standard deviation demonstrate that the number of iterations fluctuates greatly from trial to trial. The smallest values of the standard deviation - for a learning
rate around 1.0 and a threshold of 0.4 - are around the value of 250, with the number of iterations varying from 31 to 1064 of iterations. Sharp increases of the network error are acknowledged and analysed in Fujita et al. (2008) and Takase et al. (2009). They suggest these are caused by false local minima.

Fujita et al. (2008) also suggest that such problems in the learning process would affect negatively any acceleration method, as these techniques are based on inaccurate local information. In order to test this theory, the same network structure has been trained with SpikeProp with a typical value for the momentum parameter (Xin and Embrechts, 2001) under similar conditions.

Figure 6.2b shows the average number of iterations for the XOR problem when trained with SpikeProp with a momentum parameter of 0.25. The results are similar with the network trained with SpikeProp without a momentum factor. The values for the standard deviation of the number of iterations are similar for the network trained without the momentum. The difference between the
Figure 6.2: Average number of iterations (calculated over the successful trials) needed for convergence for the XOR problem for different combinations of learning rates and thresholds: (a) without, (b) with a momentum parameter
learning methods reported in Xin and Embrechts (2001) can be a consequence of a sudden rise of the network error which is present in both training conditions, as they reported only single trial results. For all the tests described to this point, the network was initialised with positive weights, with one hidden neuron being inhibitory. It has been reported (Bohte et al., 2002; Booij and Nguyen, 2005) that using mixed negative and positive weights would prevent the learning algorithm to converge. Other studies (Moore, 2002; Takase et al., 2009) reported that the learning algorithm did converge when the network was initialised with negative and positive weights.

When the network described above was initialised with positive and negative weights within a symmetric interval - (-0.5, 0.5) and (-1.0, 1.0) - the learning algorithm converged only in a maximum of 72% of times. When the network was initialised with positive and negative weights within the range (-0.1, 0.9), the learning converged 99.5% of times on average for a threshold of 0.01 and 0.005.
Figure 6.4 shows the percentage of successful learning trials when the weights were initialised within the range (-0.1, 0.9) uniformly distributed, with all neurons being excitatory. The average number of learning iterations, although slightly smaller - 197 iterations on average for the threshold 0.6 with a learning rate of 0.05 - and the standard deviation of the number of iteration needed for convergence was similar with the results for the network initialised only with positive weights and one hidden inhibitory neuron.

Figure 6.4: The percentage of successful learning trials of XOR problem with negative and positive initial weights within the range (-0.1, 0.9).

### 6.2 Reference start time

Although the gradient descent learning method has been tested with various benchmark problems, all other studies on SpikeProp (Fujita et al., 2008; McKennoch et al., 2006; Moore, 2002; Schrauwen and Van Campenhout, 2004; Takase et al., 2009; Xin and Embrechts, 2001) used the XOR problem or the Iris data set
with identical network structures as in the original article on SpikeProp (Bohte et al., 2002). This gives little insight of the capabilities or limitations of a spiking neural network trained with SpikeProp.

One such example is the topology of the neural network designed to solve the XOR problem. It was assumed by Bohte et al. (2002) that using a latency encoding (see Table 6.1), the network needs an additional input neuron to designate the reference start time, otherwise the problem becomes trivial. Indeed, since the patterns consist of the exact times of the spikes, the absolute time is irrelevant and two of the input patterns would be considered identical without the bias input neuron - the patterns that have their input neurons fire at the same time.

Consider one such network for the XOR problem, with two input neurons, five hidden neurons, and one output neuron with the same encoding of the patterns used so far (Sporea and Gruning, 2011). For the input pattern \( (t_1 = 0, t_2 = 0) \), the membrane potential of a hidden neuron \( j \) will be:

\[
x_j(t) = \sum_k w_{j1}^k \epsilon(t - t_1 - d_k) + \sum_k w_{j2}^k \epsilon(t - t_2 - d_k)
\] (6.1)

For the input pattern \( (t_1' = \Delta t, t_2' = \Delta t) \) the membrane potential of the same hidden neuron will be:

\[
x_j(s) = \sum_k w_{j1}^k \epsilon(s - t_1' - d_k) + \sum_k w_{j2}^k \epsilon(s - t_2' - d_k)
\] (6.2)

For \( s = t + \Delta t \), the above equation becomes:

\[
x_j(s) = \sum_k w_{j1}^k \epsilon(t + \Delta t - (t_1 + \Delta t) - d_k)
+ \sum_k w_{j2}^k \epsilon(t + \Delta t - (t_2 + \Delta t) - d_k)
\] (6.3)

This can be rewritten as follows:

\[
x_j(s) = \sum_k w_{j1}^k \epsilon(t - t_1 - d_k) + \sum_k w_{j2}^k \epsilon(t - t_2 - d_k)
\] (6.4)
where \( x_j(s) = x_j(t) \) with \( s = t + \Delta t \).

Thus at the time \( s = t + \Delta t \), the membrane potential has the same shape as the neuron in response to the first pattern at time \( t \). The membrane potential function \( x_j(t) \) has no "knowledge" of absolute time; hence until it receives an input \( t > 0 \), the neuron is in a passive state. When an input current arrives, the membrane potential will shape according to the input, independent of the absolute time value. This conclusion is true for any set of weights \( w_{ij} \), independent of the spike response function \( \epsilon(t) \).

This can be generalized to all neurons in the hidden layer, and to the subsequent layers. To summarise, a spiking neural network of this type will respond to a pattern of time-spikes where all the input neurons fire at the same time \( t_0 \), with a spiking time \( t_0 + \Delta T \). Thus, the same network will respond to a pattern where all input neurons fire at the same time \( t_0 + \Delta t \) with a spike time at \( t_0 + \Delta t + \Delta T \). Since this is true for any set of weights, the problem is independent of the learning algorithm. Hence, by removing the "bias" input neuron, the XOR problem not only fails to become trivial, it becomes impossible to solve (Sporea and Grünig, 2011).

Consequently, when tested the XOR problem with two input neurons \( t_i \), if presented with input patterns with identical spike times, the network responded always with \( t_i + \Delta T \), where the value of \( \Delta T \) depended on the set of weights. For all testing conditions described above, the network with only two input neurons was unable to learn the XOR function.

Moreover, when the same network structure was trained to solve the linearly separable AND function, the learning algorithm was incapable to converge. Thus if a multilayer spiking neuron network without a reference time start cannot compute linear functions such as AND, which functions can it still solve?

Table 6.2 shows all functions of two variables as an example. A spiking neural network of this sort can only compute those functions for which the response to the spike time pattern \((0, 0)\) is 10, and the response to the spike time pattern \((6, 6)\) is 16 - these are indeed trivial to compute for a network without a reference time start. There are only four functions, \( f_8, f_{10}, f_{12}, f_{14} \), that can be computed in
6. Analysis of SpikeProp

Table 6.2: Input \((t_0 - t_1)\) and output \((f_0 - f_{15})\) spike-time patterns. The patterns consists of the timing of single spikes in ms of simulated time for the input and target neurons.

| \(t_0\) | 0 | 0 | 6 | 6 |
| \(t_1\) | 0 | 6 | 0 | 6 |
| \(f_0\) | 10 | 10 | 10 | 10 |
| \(f_1\) | 16 | 10 | 10 | 10 |
| \(f_2\) | 10 | 16 | 10 | 10 |
| \(f_3\) | 16 | 16 | 10 | 10 |
| \(f_4\) | 10 | 10 | 16 | 10 |
| \(f_5\) | 16 | 10 | 16 | 10 |
| \(f_6\) | 10 | 16 | 16 | 10 |
| \(f_7\) | 16 | 16 | 16 | 10 |
| \(f_8\) | 10 | 10 | 10 | 16 |
| \(f_9\) | 16 | 10 | 10 | 16 |
| \(f_{10}\) | 10 | 16 | 10 | 16 |
| \(f_{11}\) | 16 | 16 | 10 | 16 |
| \(f_{12}\) | 10 | 10 | 16 | 16 |
| \(f_{13}\) | 16 | 10 | 16 | 16 |
| \(f_{14}\) | 10 | 16 | 16 | 16 |
| \(f_{15}\) | 16 | 16 | 16 | 16 |

this way, as opposed to a single perceptron that can compute 14 of this functions (see Rojas (1996) for a graphical and analytical demonstration).

The impossibility of the network to solve these simple logical problems depends, however, on the particular choice of encoding the logic patterns. If, for example, the output patterns are reversed (i.e. a logical output "1" is associated with a spike at 16 ms, and a logical "0" is associated with a spike at 10 ms) problems like AND do become trivial. Then again, in this case, the four problems mentioned above as trivial become impossible to solve. With this new encoding, there are still only four functions \((f_1, f_3, f_5, f_7)\) which are trivial to solve; XOR would still be impossible to solve following the same reasoning.

By considering other encodings for the output patterns, such as firing a single
spike or remaining quiescent, the XOR problem can be, in this case, learnt. Using such an encoding for the target patterns the network can compute only those functions that require the same response to the identical patterns, (0, 0) and (6, 6). However, in this case the XOR problem is no longer linearly non-separable and the benchmark tells us nothing about the network ability to learn linearly non-separable problems.

6.3 Discussion

Choosing the right set of parameters for a spiking neural network trained with SpikeProp has proved to be a rather difficult task. In order to find the optimal group of parameters and to analyse how these parameters influence the learning process, a wide range of threshold values and learning rates are considered for the XOR problem. Looking at the highest number of successful trials, the best choices of learning rate reside in the range of 0.05 and 0.01, with a maximum convergence rate of 92%. However, such low learning rates also result in a slower learning process, with more than 300 iterations on averages needed for convergence.

However, looking further at the average number of iterations needed for convergence, the optimal values for the threshold are round 0.4, with learning rates around 1.0. Examining how the number of iterations varies under the same training conditions, it can be observed that the standard deviation of the number of iterations is extremely large (see Figure 6.3). The smallest values are around 260, with the number of iteration varying between 31 and 1064. Such a high standard deviation can be explained by the network error surface which may contain sudden accelerations that increase the learning time.

Using a momentum parameter (Xin and Embrechts, 2001) in the training resulted in similar average number of iterations and its standard deviations. This is assumed to be also a consequence of the sudden increases in the network error, as this method is based on local information which can be inaccurate (Fujita et al., 2008; Takase et al., 2009). In order to improve the performance of the learning algorithm, the causes of these rapid increases of the network error need
to be identified and eliminated from the learning process (Fujita et al., 2008; Takase et al., 2009).

Although it has been reported that using both positive and negative weights, the learning algorithm would not converge (Bohte et al., 2002), when the network was initialised with mixed sign weights within the interval (-0.1, 0.9), SpikeProp always converged when a learning rate of 0.01 was used. This results are consistent with Takase et al. (2009) results which showed that the algorithm converged for weights initialised within the interval (-2, 8) with a threshold value of 20.

Another issue concerning the SpikeProp algorithm revolves around the particular choice of network configuration for the XOR function. Although it was assumed that the XOR problem would become trivial without an input neuron to designate the reference start time (Bohte et al., 2002), simulations and demonstration proved otherwise. Without the bias input neuron, the problem becomes impossible to solve for this particular encoding, independent of the learning algorithm (Sporea and Grüning, 2011).

Moreover, linear problems such as the AND function, which can be solve by a single perceptron (Minsky and Papert, 1969), also become impossible to solve. Changing the encoding of the output patterns would change the AND problem into a trivial one, but would make NAND impossible to solve. Although this limitation can be easily solved be adding a bias input neuron, other solutions can be found in different encodings. Using a learning rule that considers multiple spikes, one could designate the reference start time with an initial spike, without adding another input neuron. This would also reduce the number of weights that need to be updated during learning.
Chapter 7

New Learning Algorithm for Multilayer Spiking Neural Networks

In this chapter a new supervised learning algorithm, multilayer ReSuMe, for feed-forward spiking neural networks with hidden layers is introduced (Sporea and Grüning, 2012). The algorithm presented here overcomes some limitations of existing learning algorithms (see Section 5.2) as it can be applied to neurons firing multiple spikes and it can in principle be applied to any linearisable neuron model. The learning rule extends the ReSuMe algorithm (Ponulak and Kasiński, 2010) to multiple layers using backpropagation of the error function. The weights are updated according to STDP and anti-STDP processes and unlike SpikeProp (Bohte et al., 2002; Booij and Nguyen, 2005) can be applied to neurons firing multiple spikes in all layers. The multilayer ReSuMe is analogous to the backpropagation learning algorithm for rate neurons, while making use of spiking neurons. The proposed supervised learning algorithm combines the quality of SpikeProp, spanning to multiple layers (Bohte et al., 2002), with the flexibility of ReSuMe, which can be used with multiple spikes and with different neuron models (Ponulak and Kasiński, 2010). Part of this chapter will be published in Sporea and Grüning (2012).
7. New Learning Algorithm for Multilayer Spiking Neural Networks

In this chapter the new learning algorithm for feed-forward multilayer spiking neural networks is described. The learning rule is derived for networks with only one hidden layer, as the algorithm can be extended to networks with more hidden layers similarly. In section 7.8 the weight modifications are analysed for a simplified network with a single output neuron.

7.1 Neuron model

The input and output signals of spiking neurons are represented by the timing of spikes. A spike train is defined as a sequence of impulses fired by a particular neuron at times $t^j$. Spike trains are formalised by a sum of Dirac $\delta$ functions (Gerstner and Kistler, 2002) - equation (5.1).

In order to analyse the relation between the input and output spike trains, we use the linear stochastic neuron model. Its instantaneous firing rate $R(t)$ is formally defined as the expectation of the spike train, averaged over an infinite number of trials. An estimate of the instantaneous firing rate can be obtained by averaging over a finite number of trials (Heeger, 2000):

$$ R(t) = < S(t) > = \lim_{M \to \infty} \frac{1}{M} \sum_{j=1}^{M} S_j(t) $$

(7.1)

where $M$ is the number of trials and $S_j(t)$ is the concrete spike train for each trial. The instantaneous firing rate $R_o(t)$ of a neuron $o$ is the probability density to fire at time $t$ and is determined by the instantaneous firing rates of its presynaptic neurons $h$:

$$ R_o(t) = \frac{1}{n} \sum_{h \in \mathcal{H}} w_{oh} R_h(t) $$

(7.2)

where $n$ is the number of presynaptic neurons $h$. The weights $w_{oh}$ represent the strength of the connection between the presynaptic neurons $h$ and postsynaptic neuron $o$. The instantaneous firing rate $R(t)$ will be used for the derivation of the learning algorithm due to its smoothness and subsequently be replaced by its discontinuous estimate, the spike train $S(t)$.
7.2 Backpropagation of the network error

The learning algorithm is derived for a fully connected feed-forward network with one hidden layer. The input layer $I$ is only setting the input patterns without performing any computation on the patterns. The hidden and output layers are labelled $H$ and $O$ respectively. All neurons in one layer are connected to all neurons in the subsequent layer.

The instantaneous error function is formally defined in terms of the difference between the actual instantaneous firing rate $R^a_o(t)$ and the target instantaneous firing rate $R^d_o(t)$ for all output neurons:

$$E(t) = E(R^a_o(t)) = \frac{1}{2} \sum_{o \in O} (R^a_o(t) - R^d_o(t))^2$$  \hspace{1cm} (7.3)

In order to minimise the error function, the weights are modified using a process of gradient descent:

$$\Delta w_{oh}(t) = -\gamma \frac{\partial E(R^a_o(t))}{\partial w_{oh}}$$  \hspace{1cm} (7.4)

where $\gamma$ is the learning rate and $w_{oh}$ represents the weight between the hidden neuron $h$ and the output neuron $o$. $\Delta w_{oh}(t)$ is the weight change contribution due to the error $E(t)$ at time $t$, and the total weight change is $\Delta w = \int \Delta w(t) dt$ over the duration of the spike trains. This is analogous to the starting point of standard backpropagation for rate neurons in discrete time. For simplicity, the learning rate will be considered $\gamma = 1$ and will be suppressed in all following equations, as the step length of each learning iteration will be given by other learning parameters to be defined later on. Also, the following derivatives are understood in a functional sense.

7.2.1 Weight modifications for the output neurons

In this section the weight-update formulated for the ReSuMe learning algorithm are re-derived and connected with gradient-descent learning for linear neurons. This derivation is needed as a first step to derive our extension of ReSuMe to
7. New Learning Algorithm for Multilayer Spiking Neural Networks

subsequent layers in the next subsection. However, this derivation is more rigorous than the original derivation (Ponulak and Kasinski, 2010) and shows how ReSuMe and gradient descent are connected. It also makes Ponulak’s statement more explicit that ReSuMe can be applied to any neuron model that can, on an appropriate time scale, be approximated well enough with a linear neuron model.

As the network error is a function of the output spike train, which in turn depends on the weight \( w_{oh} \), the derivative of the error function can be expanded using the chain rule as follows:

\[
\frac{\partial E(R_o(t))}{\partial w_{oh}} = \frac{\partial E(R^a_0(t))}{\partial R^a_0(t)} \cdot \frac{\partial R^a_0(t)}{\partial w_{oh}}
\]

The first term of the right-hand part of equation (7.5) can be calculated as:

\[
\frac{\partial E(R^a_0(t))}{\partial R^a_0(t)} = R^a_0(t) - R^d_0(t)
\]

Since the instantaneous rate function is expressed in terms of the weight \( w_{oh} \) in (7.2), the second factor of the right-hand side of equation (7.5) becomes:

\[
\frac{\partial R_0(t)}{\partial w_{oh}} = \frac{1}{n_h} R_h(t)
\]

where \( n_h \) is the number of hidden neurons. By combining equations (7.4) – (7.7), the formula for weight modifications to the output neurons becomes:

\[
\Delta w_{oh}(t) = -\frac{1}{n_h} \left[ R^a_0(t) - R^d_0(t) \right] R_h(t)
\]

For convenience we define the backpropagated error \( \delta_o(t) \) for the output neuron \( o \):

\[
\delta_o(t) := \frac{1}{n_h} \left[ R^d_0(t) - R^a_0(t) \right]
\]

hence:

\[
\Delta w_{oh}(t) = \delta_o(t) R_h(t)
\]

This is similar to standard discrete-time backpropagation, however now derived as a functional derivative in continuous time. In the following we will use the
7. New Learning Algorithm for Multilayer Spiking Neural Networks

best estimation of the unknown instantaneous firing rate $R(t)$ when we only have
a single spike train $S(t)$, which is the spike train itself for each of the neurons
involved. Thus the weights will be modified according to:

$$\Delta w_{oh}(t) = \frac{1}{n_h} \left[ S_o^d(t) - S_o^a(t) \right] S_h(t) \quad (7.11)$$

However, products of Dirac $\delta$ functions are mathematically problematic. Following Ponulak and Kasiński (2010), the non-linear product of $S_o^d(t)S_h(t)$ is substituted with a STDP process. In a similar manner, $(-S_o^a(t)S_h(t))$ is substituted with an anti-STDP process (for a detailed derivation see Section 5.2.2).

$$S_o^d(t)S_h(t) \rightarrow S_h(t) \left[ a + \int_0^\infty a^{pre}(s)S_o^d(t - s)ds \right] + S_o^a(t) \left[ a + \int_0^\infty a^{post}(s)S_h(t - s)ds \right] \quad (7.12)$$

$$S_o^a(t)S_h(t) \rightarrow S_h(t) \left[ a + \int_0^\infty a^{pre}(s)S_o^a(t - s)ds \right] + S_o^d(t) \left[ a + \int_0^\infty a^{post}(s)S_h(t - s)ds \right] \quad (7.13)$$

where $a > 0$ is a non-Hebbian term that guarantees the weight changes in the
correct direction if the output spike train contains more or less spikes than the
target spike train.

The integration variable $s$ represents the time difference between the actual
firing time of the output neuron and the firing time of the hidden neuron $s = (t_o^f - t_h^f)$, and the target firing time and the firing time of the hidden neuron $s = (t_d^f - t_f^f)$ respectively. The kernel $a^{pre}(s)$ gives the weight change if the
presynaptic spike (the spike of the hidden neuron) comes after the postsynaptic
spike (the spikes of the output and target neurons). The kernel $a^{post}(s)$ gives the
weight change if the presynaptic spike before the postsynaptic spike. The kernels
7. New Learning Algorithm for Multilayer Spiking Neural Networks

$a^{\text{pre}}$ and $a^{\text{post}}$ define the learning window $W(s)$ (Gerstner and Kistler, 2002):

$$W(s) = \begin{cases} a^{\text{pre}}(-s) = -A_- \exp\left(\frac{s}{\tau_-}\right), & \text{if } s \leq 0 \\ a^{\text{post}}(s) = +A_+ \exp\left(\frac{s}{\tau_+}\right), & \text{if } s > 0 \end{cases}$$

(7.14)

where $A_+, A_- > 0$ are the amplitudes and $\tau_+, \tau_- > 0$ are the time constants of the learning window. Thus the final learning formula for the weight modifications becomes:

$$\Delta w_{ih}(t) = \frac{1}{n_h} S_h(t) \left[ \int_0^\infty a^{\text{pre}}(s)[S_o^d(t-s) - S_o^a(t-s)]ds \right]$$

$$+ \frac{1}{n_h} [S_o^d(t) - S_o^a(t)] \left[ a + \int_0^\infty a^{\text{post}}(s)S_h(t-s)ds \right]$$

(7.15)

The total weight change is obtained by integrating equation (7.15) over time on a time domain that covers all the spikes in the system. This equation is the core of ReSuMe learning algorithm as stated in Ponulak and Kasinski (2010).

7.2.2 Weight modifications for the hidden neurons

In this section we extend the argument above to weight changes between the input and the hidden layer. The weight modifications for the hidden neurons are calculated in a similar manner in the negative gradient direction:

$$\Delta w_{hi}(t) = -\frac{\partial E(R_o^a(t))}{\partial w_{hi}}$$

(7.16)

where $w_{hi}$ is the weight between input neuron $i$ and hidden neuron $h$. The derivative of the error is expanded similarly as in equation (7.5) (again in the sense of functional derivatives):

$$\frac{\partial E(R_o^a(t))}{\partial w_{hi}} = \frac{\partial E(R_o^a(t))}{\partial R_h(t)} \frac{\partial R_h(t)}{\partial w_{hi}}$$

(7.17)
The first factor of the right-hand part of the above equation is expanded for each output neuron using the chain rule:

$$\frac{\partial E(R_{o}(t))}{\partial R_{h}(t)} = \sum_{o \in O} \frac{\partial E(R_{o}(t))}{\partial R_{o}(t)} \frac{\partial R_{o}(t)}{\partial R_{h}(t)}$$  \hspace{1cm} (7.18)

The second factor of the right-hand side of the above equation is calculated from equation (7.2):

$$\frac{\partial R_{o}(t)}{\partial R_{h}(t)} = \frac{1}{n_{h}} w_{oh}$$  \hspace{1cm} (7.19)

The derivatives of the error with respect to the output spike train have already been calculated for the weights to the output neurons in equation (7.6). By combining these results:

$$\frac{\partial E(R_{o}(t))}{\partial R_{h}(t)} = \frac{1}{n_{h}} \sum_{o \in O} [R_{o}(t) - R_{o}^{d}(t)] w_{oh}$$  \hspace{1cm} (7.20)

The second factor of the right-hand part of equation (7.17) is calculated as follows using again equation (7.2):

$$\frac{\partial R_{h}(t)}{\partial w_{hi}} = \frac{1}{n_{i}} R_{i}(t)$$  \hspace{1cm} (7.21)

where $n_{i}$ is the number of input neurons. By combining equations (7.16) – (7.21), the formula for the weight modifications to the hidden neurons becomes:

$$\Delta w_{hi}(t) = -\frac{1}{n_{h}n_{i}} \sum_{o \in O} [R_{o}(t) - R_{o}^{d}(t)] R_{i}(t) w_{oh}$$  \hspace{1cm} (7.22)

We define the backpropagated error $\delta_{h}(t)$ for layers other than the output layer:

$$\delta_{h}(t) := \frac{1}{n_{i}} \sum_{o \in O} \delta_{o}(t) w_{oh}$$  \hspace{1cm} (7.23)

Just like in standard backpropagation $\delta_{o}(t)$ are backpropagated errors of the neurons in the preceding layer. By substituting the instantaneous firing rates...
with spike trains as estimators, equation (7.22) becomes:

$$\Delta w_{hi}(t) = \frac{1}{n_h n_i} \sum_{o \in O} [S_o^d(t) - S_o^a(t)] S_i(t) w_{oh}$$  (7.24)

We repeat the procedure of replacing the product of two spike trains (involving δ-distributions) with a STDP process. We note first that equation (7.24) does not depend any longer on any spikes fired or not fired in the hidden layer. While there are neurobiological plasticity processes that can convey information about a transmitted spike from the effected synapses to lateral or downstream synapses (for an overview see Harris (2008)), no direct neurobiological basis is known for an STDP process between a synapse and the outgoing spikes of an upstream neuron. Therefore this substitution is to be seen as a computational analogy, and the weights will be modified according to:

$$\Delta w_{hi}(t) = \frac{1}{n_h n_i} S_i(t) \sum_{o \in O} \left[ \int_0^\infty a^{prec}(s)[S_o^d(t - s) - S_o^a(t - s)]ds \right] w_{oh}$$

$$+ \frac{1}{n_i n_h} \sum_{o \in O} [S_o^d(t) - S_o^a(t)] \left[ a + \int_0^\infty a^{post}(s) S_i(t - s)ds \right] w_{oh}$$  (7.25)

The total weight change is again determined by integrating equation (7.25) over time. The synaptic weights between the input and hidden neurons are modified according to STDP processes between the input and target spikes and anti-STDP processes between input and output spikes.

### 7.3 Normalisation

The normalisation to the number of presynaptic connections of the modifications of the weights to the output neurons ensures that the changes are proportional to the number of weights. Moreover, the learning parameters do not need to change as the network architecture changes (for example, in order to keep the firing rate of postsynaptic neurons constant as the number of presynaptic units changes, the initial weights and weight modifications also must change accordingly). The
7. New Learning Algorithm for Multilayer Spiking Neural Networks

normalisation to the number of presynaptic and postsynaptic connections of the weight modifications to the hidden neurons ensures that the changes of the connections between the input and hidden layer are usually smaller than the changes of the connections between the hidden and output layer, which keeps the learning process stable.

7.4 Generalisation

The algorithm can be generalised using equation (7.23) for neural networks with multiple hidden layers. This is our extension of ReSuMe to hidden layers following from error minimisation and gradient descent.

As the learning rule for weight modifications depends only on the presynaptic and postsynaptic spike trains and the current strength of the connections between the spiking neurons, the algorithm can be applied to various spiking neuron models, as long as the model can be sufficiently well approximated on an appropriate time scale as in equation (7.2). Although Ponulak and Kasinski (2010) do not explicitly use any neuron model for the derivation of the ReSuMe algorithm, implicitly a linear neuron model is assumed, as this derivation shows. The ReSuMe algorithm has successfully been applied to leaky integrate-and-fire neurons, Hodgkin-Huxley, and Izhikevich neuron models (Ponulak and Kasinski, 2010). Since the present learning rule is an extension of ReSuMe to neural networks with multiple layers, this is an indication that this algorithm will function with similar neuron models, as we demonstrate in the following chapter.

7.5 Inhibitory connections

Inhibitory connections are represented by negative weights which are updated in the same manner as positive weights. However, for the calculation of the backpropagation error of the hidden neurons \( \delta_h(t) \) in equation (7.25), the absolute value of the output weights will be used. This is a deviation from the gradient
7. New Learning Algorithm for Multilayer Spiking Neural Networks

descent rule, but using the absolute values guarantees that the weights between the input and hidden neurons are always modified in the same direction as between hidden and output neurons:

\[
\Delta w_{hi}(t) = \frac{1}{n_in_h} S_i(t) \sum_{o \in O_o} \left[ \int_{0}^{\infty} a^{pre}[S^d_o(t) - S^a_o(t)] \right] |w_{oh}|
\]

\[
+ \frac{1}{n_in_h} \sum_{o \in O_o} \left[ S^d_o(t) - S^a_o(t) \right] \left[ a + \int_{0}^{\infty} a^{post}(s) S_i(t - s) ds \right] |w_{oh}|.
\]

(7.26)

Preliminary simulations have shown this results in better convergence of the learning algorithm. There is also neurobiological evidence that LTD and LTP spread to downstream synapses (Fitzsimonds et al., 1997; Tao et al., 2000), i.e. that weight changes with the same direction propagate from upstream to downstream neurons.

7.6 Delayed sub-connections

If one considers a network architecture where all the neurons in one layer are connected to all neurons in the subsequent layer through multiple sub-connections with different delays \( d^k \), where each sub-connection has a different weight (Bohte et al., 2002), the learning rule for the weight modifications for the output neurons will become:

\[
\Delta w_{oh}^k = \delta_o(t) R_h(t - d_{oh}^k)
\]

(7.27)

where \( w_{oh}^k \) is the weight between output neuron \( o \) and hidden neuron \( h \) delayed by \( d_{oh}^k \) ms. The backpropagated error for the output is then:

\[
\delta_o(t) = \frac{1}{mn_h} \left[ R^d_o(t) - R^a_o(t) \right]
\]

(7.28)
7. New Learning Algorithm for Multilayer Spiking Neural Networks

where \( m \) is the number of sub-connections. The learning rule for the weight modifications for any hidden layer is derived similarly as:

\[
\Delta w_{hi}^k = \delta_h(t)R_i(t - d_{oh}^k)
\]  

(7.29)

where \( \delta_h(t) \) is the backpropagated error calculated over all possible backward paths (from all output neurons through all delayed sub-connections):

\[
\delta_h(t) = \frac{1}{mn_i} \sum_{t_o \in O} \delta_o w_{oh}^t
\]  

(7.30)

The algorithm can be generalised for neural networks with multiple hidden layers and delays similarly.

7.7 Synaptic scaling

There has been extensive evidence that suggests that spike-timing dependent plasticity is not the only form of plasticity (Watt and Desai, 2010). Another plasticity mechanism used to stabilise the neurons activity is synaptic scaling (Shepard et al., 2009). Synaptic scaling regulates the strength of synapses in order to keep the neuron’s firing rate within a particular range. The synaptic weights are scaled multiplicatively, this way maintaining the relative differences in strength between any inputs (Watt and Desai, 2010).

In our network, in addition to the learning rule described above, the weights are also modified according to synaptic scaling in order to keep the postsynaptic neuron firing rate within an optimal range \([r_{min}, r_{max}]\). If a weight \( w_{ij} \) from neuron \( j \) to neuron \( i \) causes the postsynaptic neuron to fire with a rate outside the optimal range, the weights are scaled according to the following formula (Grüning and Sporea, 2012):

\[
w_{ij} = \begin{cases} 
(1 + f)w_{ij}, & w_{ij} > 0 \\
\frac{1}{1+f}w_{ij}, & w_{ij} < 0 
\end{cases}
\]  

(7.31)
7. New Learning Algorithm for Multilayer Spiking Neural Networks

where the scaling factor $f > 0$ for $r_i < r_{\text{min}}$, and $f < 0$ for $r_i > r_{\text{max}}$, with $r_i$ the firing rate of the postsynaptic neuron.

Synaptic scaling solves the problem of optimal weight initialisation. It was observed that the initial values of the weights have a significant influence on the learning process, as too large or too low values may result in failure of the learning (see Section 6.1). Experiments show that a feed-forward network can still learn reliably simple spike trains without synaptic scaling as long as the weights are initialised within an optimal range. However, as the target patterns contain more spikes, finding the optimal initial values for the weights becomes difficult. Moreover, as the firing rate of the target neurons increases, it becomes harder to maintain the output neurons firing rate within the target range without using minimal learning steps. The introduction of synaptic scaling solves the problem of weight initialisation as well as speeds up the learning process as shown by preliminary experiments.

7.8 Heuristic motivation of the learning rule

In order to analyse the direction in which the weights change during the learning process using equations (7.15) and (7.26), we will consider a simple three layer network. The output layer consists of a single neuron. The neurons are connected through a single sub-connection with no delay. For clarity, in this section spike trains will comprise only a single spike. Let $t_d$ and $t_o$ denote the desired and actual spike time of output neuron $o$, $t_h$ and $t_i$ the spikes times of the hidden neuron $h$ and input neuron $i$ respectively. Also, for simplicity, synaptic scaling will not be considered here.

For a start we assume $t_o, t_d > t_h > t_i$, i.e. where relevant postsynaptic spikes occur after the presynaptic spikes. With these assumptions (7.15) and (7.26)
7. New Learning Algorithm for Multilayer Spiking Neural Networks

become after integrating over time:

$$\Delta w_{oh} = \frac{1}{n_h} \left( aN + A_+ \exp \left( \frac{t_h - t_d}{\tau_+} \right) - A_+ \exp \left( \frac{t_h - t_o}{\tau_+} \right) \right),$$  \hspace{1cm} (7.32)

$$\Delta w_{hi} = \frac{1}{n_h n_i} |w_{oh}| \left( aN + A_+ \exp \left( \frac{t_i - t_d}{\tau_+} \right) - A_+ \exp \left( \frac{t_i - t_o}{\tau_+} \right) \right).$$  \hspace{1cm} (7.33)

where $N$ is the difference between the spike count of the actual output spike train and the target spike train.

In the following we discuss all cases for $t_o, t_d > t_h > t_i$, and note that the case $t_o, t_d < t_h, t_i$ (i.e. post-before-pre) can be discussed along the same lines with $A_+$ above replaced by $A_-$. 

1. The output neuron fires a spike at time $t_o$ before the target firing time $t_d$ ($t_o < t_d$).

(a) **Weight modifications for the synapses between the hidden and output neurons.** The weights are modified according to $\Delta w_{oh} = \frac{1}{n_h} (A_+ \exp \frac{t_h - t_d}{\tau_+} - A_+ \exp \frac{t_h - t_o}{\tau_+})$. Since $t_o < t_d$ then $\exp \left( \frac{t_h - t_o}{\tau_+} \right) > \exp \left( \frac{t_h - t_d}{\tau_+} \right)$ in equation (7.32). This results in $\Delta w_{oh} < 0$, and thus in a decrease of this weight. If the connection is an excitatory one, the connection becomes less excitatory, increasing the likelihood that the output neuron fires later during the next iteration, hence minimising the difference between the actual output and the target firing time. If the connection is inhibitory, the connection will be strengthened, increasing the inhibition to the output neuron and resulting in a later firing of the output neuron $o$ as well (see also Ponulak (2006)).

(b) **Weight modifications for the synapses between the input and hidden neurons.** The weights to the hidden neurons are modified according to: $\Delta w_{hi} = \frac{1}{n_h n_i} (A_+ \exp \frac{t_i - t_d}{\tau_+} - A_+ \exp \frac{t_i - t_o}{\tau_+})|w_{oh}|$.

(i) $w_{oh} \geq 0$. By an analogous reasoning to the case above $\Delta w_{hi} < 0$, and hence the connection will become less excitatory or more inhibitory, again making the hidden neuron fire slightly later, and
7. New Learning Algorithm for Multilayer Spiking Neural Networks

hence making it more likely that the output neuron will fire later as the connection from hidden to output layer is excitatory.

(ii) $w_{oh} < 0$. For the hidden neuron the effect stays the same, hence it will fire later. As it is now more likely to fire later, its inhibitory effect will come to bear on the output neuron to also fire slightly later.

2. The output neuron fires a spike at time $t_o$ after the target firing time $t_d$ ($t_o > t_d$).

(a) Weight modifications for the synapses between the hidden and output neurons. Since $t_o > t_d$ then $\exp \left( \frac{t_h - t_o}{\tau_h} \right) < \exp \left( \frac{t_h - t_d}{\tau_h} \right)$ in equation (7.32). This results in $\Delta w_{oh} > 0$, and an increase of the weight. If the connection is an excitatory one, the connection is strengthened causing the output neuron to fire sooner and thus minimising the difference between the actual output and the target firing time. If the weight is negative, the inhibition to the output neuron will be weakened, making the output neuron to trigger an action potential sooner.

(b) Weight modifications for the synapses between the input and hidden neurons.

(i) $w_{oh} \geq 0$. Again, by an analogous reasoning to the case above $\Delta w_{hi} > 0$, and hence the connection will become more excitatory or less inhibitory, again making the hidden neuron fire a sooner, and hence making it more likely that also the output neuron will fire sooner.

(ii) $w_{oh} < 0$. For the hidden neuron the effect stays the same, hence the hidden neuron will fire sooner. As it is now more likely to fire sooner, the inhibitory effect of the output synapse will cause the output neuron to fire sooner.

3. The neuron fires a spike at time $t_o$ although the target spike train contains no spike, which results in $N = -1$. We may assume $t_d$ approaches infinity (see also (Ponulak, 2006)).
7. New Learning Algorithm for Multilayer Spiking Neural Networks

(a) Weight modifications for the synapses between the hidden and output neurons. The weights are updated according to $\Delta w_{oh} = \frac{1}{n_h}(-a - A_+ \exp\frac{b_h - t_h}{\tau_h})$, where the parameter $a$ is given by the difference between the spike count of the target and actual output spike trains. The change of the weight to the output neuron will always be a negative number. In the case of a positive weight, this is decreased so that the total excitation would not reach the threshold. If the weight is negative, the inhibitory connection is strengthened in order to reduce the output neuron's ability to trigger a spike.

(b) Weight modifications for the synapses between the input and hidden neurons. The weights are updated according to $\Delta w_{oi} = \frac{1}{n_i}(-a - A_+ \exp\frac{b_i - t_i}{\tau_i})|w_{oi}|$

(i) $w_{oi} \geq 0$. Again, by an analogous reasoning to the case above $\Delta w_{hi} < 0$, and hence the connection will become less excitatory or more inhibitory, again making the hidden neuron fire later, and hence making it more likely that also the output neuron will fire later.

(ii) $w_{oi} < 0$. For the hidden neuron the effect stays the same, hence the hidden neuron will fire later. As it is now more likely to fire later, its inhibitory effect will reduce the output neuron's chances to fire a spike.

4. The output neuron does not fires an action potential although the target spike train contains a spike at time $t_d$, which results in $N = 1$. We may assume that $t_o$ approaches infinity as in (Ponulak, 2006).

(a) Weight modifications for the synapses between the hidden and output neurons. In this case, $\Delta w_{oh} = a + A_+ \exp\frac{b_h - t_h}{\tau_h}$, where $a$ is given by the difference between the spike count of the target and actual output spike trains. The change of the weight to the output neuron will always be a positive number. In the case of a positive weight, the connection is strengthened so that the total excitation of the output neuron would reach the threshold and increasing the probability of the output neuron
7. New Learning Algorithm for Multilayer Spiking Neural Networks

to fire a spike. If the weight is negative, the inhibitory connection is weaken so that the output neuron will trigger a spike.

(b) Weight modifications for the synapses between the input and hidden neurons. The weights are modified according to \( \Delta w_{hi} = \frac{1}{n_h} (a + A_+ \exp \left( \frac{t-t_d}{\tau_h} \right)) |w_{oh}|. \)

(i) \( w_{oh} \geq 0. \) Again, by an analogous reasoning to the case above \( \Delta w_{hi} > 0, \) and hence the connection will become more excitatory or less inhibitory, again making the hidden neuron fire sooner, and hence making it more likely that also the output neuron will fire.

(ii) \( w_{oh} < 0. \) For the hidden neuron the effect stays the same, hence it will fire sooner. As it is now more likely to fire sooner, its inhibitory effect will cause the output neuron to fire a spike.

7.9 Summary

In this chapter, a new learning algorithm for feed-forward networks of spiking neurons is introduced. Unlike existing supervised learning rules, such as Spike-Prop and its extensions described in Section 5.2.3, multilayer ReSuMe assumes a linear neuron model and thus, it can in principle be applied to any linearisable neuron model. Moreover, multiple spikes are allowed in all layers. The algorithm extends ReSuMe to networks with hidden layers, using the well-known paradigm of gradient descent.
Chapter 8

Simulations

In this chapter several experiments are presented to illustrate the learning capabilities of multilayer ReSuMe. The algorithm is applied to classic benchmarks, the XOR problem and the Iris data set, as well as to classification tasks with randomly generated patterns. The XOR problem is applied using two different coding schemes to demonstrate the flexibility of our learning algorithm. The learning rule is also applied to classification and mapping problems of spike timing patterns which range from 100 ms to 500 ms in order to simulate sensory and motor processing in biological systems. Details of the implementation of the algorithm are given in Appendix B. Part of this chapter will be published in Sporea and Grüning (2012).

8.1 Experimental setup

For all simulations, an iteration consists of presenting all spike timing pattern pairs in a random order. The membrane potential of all neurons in the hidden and output layers is set to the resting potential (set to zero) when presenting a new input pattern. After the presentation of each input pattern to the network, the weight modifications are computed for all layers and then applied. We apply the weight changes after the backpropagated error is computed for all units in
the network. The summed network error is calculated for all patterns and tested against a required minimum value, depending on the experiment. This minimum value is chosen in order to guarantee that the network has learnt to correctly classify all the patterns with an acceptable precision.

The results are averaged over a large number of trials (50 trials unless stated otherwise), with the network being initialised with a new set of random weights every trial. On each testing trial the learning algorithm is applied for a maximum of 2000 iterations or until the network error has reached the minimum value.

The learning is considered converged if the network error has reached a minimum value, depending on the experiment. Additional constrains for the convergence of the learning algorithm are considered in Sections 8.4 to 8.6 in order to ensure the network has learnt to correctly classify all the patterns. For all simulations, the average number of iterations needed for convergence is calculated over the successful trials. Again, for all simulations, the average number of iterations is shown along the standard error of the mean. Additionally, for all simulations the output signals during the learning process for a sample trial are illustrated. The accuracy rate is defined as the percentage of correctly classified patterns calculated over the successful trials.

8.1.1 Neuron model

The network used for the following simulations is a feed-forward architecture with three layers. The computing units of the feed-forward network used in all simulations are described by the Spike Response Model (for a detailed description see Section 4.2.2).

The emission of an action potential is described by a threshold process as follows. The spike is triggered if the membrane potential $u(t)$ of a neuron reaches the threshold $\vartheta$ at time $t^f$ (see equation (4.20)). The neuron is characterised by a single variable, the membrane potential, $u(t)$ at time $t$.

In the case of a single neuron $j$ receiving input from a set of presynaptic
neurons $i \in \Gamma_j$, the state of the neuron is described as follows:

$$u_j(t) = \eta(t - t_j^f) + \sum_{i \in \Gamma_j} \sum_k w_{ji} y_i$$

(8.1)

where $y_i$ is the spike response function of the presynaptic neuron $i \in \Gamma_j$, and $w_{ji}$ is the weight between neurons $i$ and $j$; $t_j^f$ is the last firing time of neuron $j$. The kernel $\eta(t)$ includes the form of the action potential as well as the after-potential:

$$\eta(t) = -\vartheta \exp \left( -\frac{t}{\tau_\vartheta} \right)$$

(8.2)

where $\tau_\vartheta > 0$ is the membrane time constant, with $\eta(t) = 0$ for $t \leq 0$.

The unweighted contribution of a single synaptic to the membrane potential is given by:

$$y_i^k(t) = \sum_f \varepsilon \left( t - t_i^f \right)$$

(8.3)

with $\varepsilon(t)$ is the spike response function with $\varepsilon(t) = 0$ for $t \leq 0$. The times $t_i^f$ represent the firing times of neuron $i$. In our case the spike response function $\varepsilon(t)$ describes a standard post-synaptic potential:

$$\varepsilon(t) = \frac{t}{\tau} \exp \left( 1 - \frac{t}{\tau} \right), \text{ for } t > 0$$

(8.4)

where $\tau > 0$ models the membrane potential time constant and determines the rise and decay of the function.

### 8.1.2 Network error

The network error for one pattern is defined in terms of the van Rossum distance between each output spike train and each target spike train (van Rossum, 2001). The error between the target spike train and the actual spike train is defined as the Euclidean distance of the two filtered spike trains (van Rossum, 2001). The
filtered spike train is determined by an exponential function associated with the spike train:

\[ f(t) = \sum \exp[-(t - t_i)/\tau_c]H(t - t_i) \]  

(8.5)

where \( t_i \) are the times of the spikes, and \( H(t) \) is the Heaviside function. \( \tau_c \) is the time constant of the exponential function. \( \tau_c \) is chosen to be appropriate to the inter spike interval of the output neurons (van Rossum, 2001). In the following simulations the output neurons are required to fire approximately one spike in 10 ms, thus \( \tau_c = 10 \) ms. The distance between two spike trains is the squared Euclidean distance between these two functions:

\[ D^2(f, g) = \frac{1}{\tau_c} \int_0^{T_c} [f(t) - g(t)]^2 \, dt \]  

(8.6)

where the distance is calculated over a time domain \([0, T]\) that covers all the spikes in the system. The van Rossum distance is also used to determine the output pattern during learning and testing. The output pattern is determined as the closest to one of the target patterns in terms of the van Rossum distance.

### 8.1.3 Learning and network parameters

Unless stated otherwise, the network parameters used in these simulations are: the threshold \( \vartheta = 0.7 \), the time constant of the spike response function \( \tau = 7 \) ms, the time constant of after-potential kernel \( \tau_r = 12 \) ms. The scaling factor is set to \( f = \pm 0.005 \) (see equation 7.31). The learning parameters are initialised as follows: \( A_+ = 1.2, A_- = 0.5, \tau_+ = \tau_- = 5 \) ms, \( a = 0.05 \).

The weights were initialised with random values uniformly distributed between -0.2 and 0.8. The weights are then normalised by dividing them to the total number of sub-connections.
8.2 The XOR benchmark

In order to demonstrate and analyse the new learning rule, the algorithm is applied to the XOR problem. While this benchmark does not require generalising, the XOR logic gate is a linearly non-separable problem and it is a classical benchmark for testing the learning algorithm's ability to train non-trivial input output transformations (Rojas, 1996).

8.2.1 Technical details

The input and output patterns are encoded using spike-time patterns as in Section 6.1 – see Table 6.1. We also used a third input neuron that designates the reference start time as this encoding needs an absolute reference start time to determine the latency of the firing (see Section 6.2). Without a reference start time, two of the input patterns become identical and without an absolute reference time, the network is unable to distinguish the two patterns (0-0 and 6-6) and would always respond with a delayed output.

The learning algorithm is applied to a feed-forward network as described above. The input layer is composed of three neurons, the hidden layer contains five spiking neurons, and the output layer contains only one neuron. Multiple sub-connections with different delays were used for each connection in the spiking neural network. Experiments showed that 12 sub-connections with delays from 0 ms to 11 ms are sufficient to learn the XOR problem (see Section 8.2.4). The results are averaged over 100 trials. The network error is summed over all pattern pairs, with a minimum value for convergence of 0.2. The minimum value is chosen to ensure that the network has learnt to classify all patterns correctly, by matching the exact number of spikes of the target spike train as well as the timing of the spikes with 1 ms precision. Each spiking neuron in the network was simulated for a time window of 30 ms, with a time step of 0.1 ms. In the following we systematically vary the parameters of the learning algorithm and examine their effects.
8.2.2 The learning parameters

Here, we vary the learning parameters $A_+$ and $A_-$ in equations (7.15) and (7.26) in order to determine the most appropriate values. $A_+$ is varied between 0.5 and 2.0, while keeping $A_- = \frac{1}{2}A_+$. Table 8.1a shows the summarised results.

Table 8.1: Summarised results for the XOR problem: (a) The parameters $A_+$ and $A_-$ are varied in order to determine the best values for faster convergence. The ratio between these parameters is constant $A_+ = 2A_-$. (b) While keeping $A_+ = 1.2$ fixed, $A_-$ is varied in order to determine the best ratio between these parameters.

<table>
<thead>
<tr>
<th>$A_+$</th>
<th>Successful trials [%]</th>
<th>Average number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>97</td>
<td>331 ± 46</td>
</tr>
<tr>
<td>0.6</td>
<td>98</td>
<td>232 ± 24</td>
</tr>
<tr>
<td>0.7</td>
<td>95</td>
<td>262 ± 38</td>
</tr>
<tr>
<td>0.8</td>
<td>97</td>
<td>144 ± 35</td>
</tr>
<tr>
<td>0.9</td>
<td>96</td>
<td>184 ± 23</td>
</tr>
<tr>
<td>1.0</td>
<td>97</td>
<td>204 ± 34</td>
</tr>
<tr>
<td>1.1</td>
<td>94</td>
<td>166 ± 27</td>
</tr>
<tr>
<td>1.2</td>
<td>96</td>
<td>207 ± 31</td>
</tr>
<tr>
<td>1.3</td>
<td>95</td>
<td>174 ± 30</td>
</tr>
<tr>
<td>1.4</td>
<td>97</td>
<td>183 ± 28</td>
</tr>
<tr>
<td>1.5</td>
<td>93</td>
<td>204 ± 36</td>
</tr>
<tr>
<td>1.6</td>
<td>93</td>
<td>273 ± 43</td>
</tr>
<tr>
<td>1.7</td>
<td>96</td>
<td>163 ± 27</td>
</tr>
<tr>
<td>1.8</td>
<td>94</td>
<td>181 ± 32</td>
</tr>
<tr>
<td>1.9</td>
<td>95</td>
<td>221 ± 32</td>
</tr>
<tr>
<td>2.0</td>
<td>89</td>
<td>141 ± 18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_-$</th>
<th>Successful trials [%]</th>
<th>Average number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>97</td>
<td>231 ± 30</td>
</tr>
<tr>
<td>0.10</td>
<td>98</td>
<td>196 ± 24</td>
</tr>
<tr>
<td>0.20</td>
<td>96</td>
<td>157 ± 16</td>
</tr>
<tr>
<td>0.30</td>
<td>96</td>
<td>187 ± 28</td>
</tr>
<tr>
<td>0.40</td>
<td>95</td>
<td>204 ± 37</td>
</tr>
<tr>
<td>0.50</td>
<td>98</td>
<td>137 ± 16</td>
</tr>
<tr>
<td>0.60</td>
<td>96</td>
<td>207 ± 31</td>
</tr>
<tr>
<td>0.70</td>
<td>95</td>
<td>191 ± 33</td>
</tr>
<tr>
<td>0.80</td>
<td>98</td>
<td>185 ± 31</td>
</tr>
<tr>
<td>0.90</td>
<td>86</td>
<td>203 ± 31</td>
</tr>
<tr>
<td>1.00</td>
<td>88</td>
<td>200 ± 30</td>
</tr>
<tr>
<td>1.10</td>
<td>80</td>
<td>257 ± 33</td>
</tr>
<tr>
<td>1.20</td>
<td>70</td>
<td>349 ± 42</td>
</tr>
<tr>
<td>1.30</td>
<td>65</td>
<td>382 ± 30</td>
</tr>
<tr>
<td>1.40</td>
<td>45</td>
<td>353 ± 28</td>
</tr>
<tr>
<td>1.50</td>
<td>56</td>
<td>492 ± 32</td>
</tr>
</tbody>
</table>

The parameters $A_+$ and $A_-$ play the role of a learning rate. Just like the classic back-propagation algorithm for rate neurons, when the main learning parameter, $A_+$, has higher values the number of iterations needed for convergence is lower.

In order to determine the best ratio between the two learning parameters, various values are chosen for $A_-$, while keeping $A_+ = 1.2$ fixed. The results
are summarised in Table 8.1b. The learning algorithm is able to converge for the values of $A_-$ lower than $A_+$. As $A_-$ becomes equal or higher than $A_+$, the convergence rate slowly decreases and the number of iterations needed for convergence significantly rises. The lowest average number of iterations with a high convergence rate is 137 averaged over 98% successful trials (for $A_+ = 1.2$ and $A_- = 0.5$).

8.2.3 Comparison with SpikeProp

In order to make a direct comparison with SpikeProp, the stopping criterion is set in terms of the time difference between the first output spike and the target spike. Table 8.2a shows the results where $A_-$ is varied with different values and $A_+ = 1.2$. Table 8.2b shows the results for a network trained with SpikeProp, with the threshold is set to 0.4, and the learning rate is varied with different values (the weights are initialised with random values between -0.2 and 0.8; the rest of the learning and network parameters are set as in section 6.1). For both networks, the learning stops when the network error (as defined in Section 6.1) reaches the value 1 ms.

The multilayer ReSuMe converges faster than SpikeProp in all cases. Also, the standard error is lower for the multilayer ReSuMe suggesting a more stable learning process.

8.2.4 Number of sub-connections

The algorithm also converges when the spiking neural network has a smaller number of sub-connections. However, a lower number of delayed sub-connections results in a lower convergence rate without necessarily a lower average of learning iterations for the successful trials. Although more sub-connections can produce a more stable learning process, due to the larger number of weights that need to be coordinated, the learning process is slower in this case. Table 8.3 shows the summarised results, where $A_+ = 1.2$ and $A_- = 0.6$. 
8. Simulations

Table 8.2: Summarised results for the XOR problem: (a) The network is trained with multilayer ReSuMe, where the learning stops when the time difference between the actual output spike and target spike is 1 ms; $A_+$ is set to 1.2, and $A_-$ is varied with different values (b) The network is trained with SpikeProp; again the learning stops when the time difference between the actual output spike and target spike is 1 ms.

<table>
<thead>
<tr>
<th>$A_-$</th>
<th>Successful trials [%]</th>
<th>Average number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>100</td>
<td>140 ± 11</td>
</tr>
<tr>
<td>0.10</td>
<td>100</td>
<td>163 ± 16</td>
</tr>
<tr>
<td>0.20</td>
<td>100</td>
<td>136 ± 13</td>
</tr>
<tr>
<td>0.30</td>
<td>100</td>
<td>143 ± 15</td>
</tr>
<tr>
<td>0.40</td>
<td>100</td>
<td>206 ± 38</td>
</tr>
<tr>
<td>0.50</td>
<td>100</td>
<td>152 ± 20</td>
</tr>
<tr>
<td>0.60</td>
<td>100</td>
<td>160 ± 19</td>
</tr>
<tr>
<td>0.70</td>
<td>100</td>
<td>168 ± 35</td>
</tr>
<tr>
<td>0.80</td>
<td>100</td>
<td>167 ± 31</td>
</tr>
<tr>
<td>0.90</td>
<td>100</td>
<td>190 ± 31</td>
</tr>
<tr>
<td>1.00</td>
<td>100</td>
<td>154 ± 16</td>
</tr>
<tr>
<td>1.10</td>
<td>100</td>
<td>157 ± 19</td>
</tr>
<tr>
<td>1.20</td>
<td>100</td>
<td>158 ± 35</td>
</tr>
<tr>
<td>1.30</td>
<td>100</td>
<td>130 ± 15</td>
</tr>
<tr>
<td>1.40</td>
<td>100</td>
<td>178 ± 28</td>
</tr>
<tr>
<td>1.50</td>
<td>100</td>
<td>188 ± 36</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Learning rate</th>
<th>Successful trials [%]</th>
<th>Average number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td>96</td>
<td>2412 ± 120</td>
</tr>
<tr>
<td>0.002</td>
<td>100</td>
<td>1434 ± 112</td>
</tr>
<tr>
<td>0.004</td>
<td>100</td>
<td>1001 ± 91</td>
</tr>
<tr>
<td>0.006</td>
<td>100</td>
<td>575 ± 43</td>
</tr>
<tr>
<td>0.008</td>
<td>100</td>
<td>590 ± 58</td>
</tr>
<tr>
<td>0.01</td>
<td>100</td>
<td>615 ± 60</td>
</tr>
<tr>
<td>0.02</td>
<td>100</td>
<td>392 ± 33</td>
</tr>
<tr>
<td>0.04</td>
<td>100</td>
<td>326 ± 32</td>
</tr>
<tr>
<td>0.06</td>
<td>100</td>
<td>222 ± 27</td>
</tr>
<tr>
<td>0.08</td>
<td>96</td>
<td>273 ± 38</td>
</tr>
<tr>
<td>0.1</td>
<td>98</td>
<td>378 ± 67</td>
</tr>
<tr>
<td>0.2</td>
<td>98</td>
<td>510 ± 126</td>
</tr>
<tr>
<td>0.4</td>
<td>82</td>
<td>245 ± 54</td>
</tr>
<tr>
<td>0.6</td>
<td>72</td>
<td>433 ± 115</td>
</tr>
<tr>
<td>0.8</td>
<td>74</td>
<td>540 ± 137</td>
</tr>
<tr>
<td>1.0</td>
<td>56</td>
<td>753 ± 177</td>
</tr>
</tbody>
</table>

8.2.5 Analysis of the learning process

In order to analyse the learning process, the network error and the weight vector during the learning process for a sample trial can be seen in Figure 8.1 ($A_+ = 1.2$, $A_- = 0.6$, and 12 sub-connections). Figure 8.1a shows the evolution of the summed network error during learning. Although the network error reaches a minimum value after 63 iterations, due to the nature of the STDP processes, the
8. Simulations

Table 8.3: The number of delayed sub-connections is varied while keeping the learning parameters fixed $A_+ = 1.2$ and $A_- = 0.6$.

<table>
<thead>
<tr>
<th>Sub-connections</th>
<th>Successful trials [%]</th>
<th>Average number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>11</td>
<td>63 ± 20</td>
</tr>
<tr>
<td>6</td>
<td>24</td>
<td>169 ± 37</td>
</tr>
<tr>
<td>8</td>
<td>73</td>
<td>192 ± 27</td>
</tr>
<tr>
<td>10</td>
<td>81</td>
<td>154 ± 17</td>
</tr>
<tr>
<td>12</td>
<td>96</td>
<td>207 ± 31</td>
</tr>
<tr>
<td>14</td>
<td>96</td>
<td>309 ± 52</td>
</tr>
<tr>
<td>16</td>
<td>73</td>
<td>472 ± 56</td>
</tr>
</tbody>
</table>

solution is lost, only to converge again later. Similar findings also were reported in Grüning and Sporea (2012).

Figure 8.1b shows the Euclidean distance between the weight vector solution found on this particular trial and the weight vectors during each learning iteration that led to this weight vector. The weight vectors are tested against the solution found during this particular trial because there can be multiple weight vectors solutions. While the error graph is irregular, the weight vector graph shows that the weight vector moves steadily towards the solution. The irregularity of the network error during the learning process can be explained by the fact that small changes to the weights can produce additional or missing output spikes, which cause significant changes in the network error. The highest error value corresponds to the network not firing any spike for any of the four input patterns. The error graph also shows the learning rule ability to modify the weights in order to produce the correct number of output spikes. Figure 8.1c shows the output signals during learning for all input patterns. Figure 8.1d shows the hidden signals during learning for each of the hidden neurons for one of the patterns.
Figure 8.1: Analysis of the learning process for the XOR benchmark with the parameters $A_+ = 1.2$ and $A_- = 0.5$: (a) The network error during learning. (b) The Euclidean distance between the weight vector solution and the weight vectors during the learning process. (c) The output signals during learning for each of the four patterns. The x markers represent the target spike times. (d) The hidden signals during learning for each hidden neuron for one input pattern ([0 0]).
8. Simulations

8.3 The *Iris* benchmark

Another classic benchmark of pattern recognition is Fisher's *Iris* flower data set (Fisher, 1936). The data set contains three classes of *Iris* flowers. While one of the classes is linearly separable from the other two, the other two classes are not linearly separable from each other.

8.3.1 Technical details

The three species are completely described by four measurements of the plants: the lengths and widths of the petal and sepal. Each of the four features is associated with the timing of a single spike of four input neurons. The measurements of the *Iris* flower range from 0 to 8 (as seen in Table 8.4) and are fed into the spiking neural network as spike timing patterns to the input neurons. The output of the network is represented by the spike-time of the output neuron, as seen in Table 8.4. The hidden layer contains ten spiking neurons and each connection has between 8 and 12 delayed sub-connections depending on the experiment. The network is simulated in a 30 ms time window with 0.1 ms time step.

Table 8.4: The input and target patterns contain a single spike, where the timing (shown in ms) differs for each of the three patterns.

<table>
<thead>
<tr>
<th>Species</th>
<th>Sepal length range [ms]</th>
<th>Sepal width range [ms]</th>
<th>Petal length range [ms]</th>
<th>Petal width range [ms]</th>
<th>Output [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>setosa</em></td>
<td>4.3-5.8</td>
<td>2.3-4.8</td>
<td>1.0-1.9</td>
<td>0.1-0.6</td>
<td>10</td>
</tr>
<tr>
<td><em>versicolor</em></td>
<td>4.9-7.0</td>
<td>2.0-3.4</td>
<td>3.0-5.1</td>
<td>1.0-1.8</td>
<td>14</td>
</tr>
<tr>
<td><em>virginica</em></td>
<td>4.9-7.9</td>
<td>2.2-3.8</td>
<td>4.5-6.9</td>
<td>1.4-2.5</td>
<td>18</td>
</tr>
</tbody>
</table>

During each trial, the input patterns are randomly divided into a training set (75% of samples) and a testing set (25% of samples) for cross validation. During each iteration, the training set is used for the learning process to calculate the weight modifications and to evaluate if the network has learnt the patterns. The learning is considered successful if the network error has reached a minimum
8. Simulations

average value of 0.2 for each pattern pair and 95% of the patterns in the training set are correctly classified. As in the previous experiment, this minimum value is chosen to ensure that the network has learnt to classify all patterns correctly, by matching the exact number of spikes of the target spike train as well as timing of the spikes with 1 ms precision. Figure 8.2 shows the output signals during learning for all three classes of species during a sample trial. While the first pattern is learnt after only a few iterations, it takes more than 100 iterations to learn all three pattern classes. Table 8.5 shows the summarised results on the Iris data set for different network architectures with different numbers of delayed sub-connections.

![Figure 8.2: Output signals during learning for the Iris data set for all input patterns for a sample trial. The x markers represent the target spike times.](image)

Table 8.5: Summarised results for the Iris data set.

<table>
<thead>
<tr>
<th>Sub-connections</th>
<th>Successful trials [%]</th>
<th>Average number of iterations</th>
<th>Accuracy on the training set [%]</th>
<th>Accuracy on the testing set [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>68</td>
<td>125 ± 12</td>
<td>97 ± 0.17</td>
<td>89 ± 0.69</td>
</tr>
<tr>
<td>9</td>
<td>80</td>
<td>174 ± 16</td>
<td>96 ± 0.00</td>
<td>94 ± 0.79</td>
</tr>
<tr>
<td>10</td>
<td>80</td>
<td>114 ± 13</td>
<td>97 ± 0.00</td>
<td>89 ± 0.47</td>
</tr>
<tr>
<td>11</td>
<td>74</td>
<td>140 ± 15</td>
<td>96 ± 0.16</td>
<td>86 ± 0.49</td>
</tr>
<tr>
<td>12</td>
<td>68</td>
<td>183 ± 21</td>
<td>96 ± 0.17</td>
<td>91 ± 0.69</td>
</tr>
</tbody>
</table>

Multilayer ReSuMe permits the spiking neural network to learn the Iris data set using a straightforward encoding of the patterns and results in much faster
learning than SpikeProp, as the average number of iterations is always lower than 200, as opposed to the population coding based on arrays of receptive fields that requires 1000 iterations for learning (Bohte et al., 2002).

8.4 Non-linear spike train pattern classification

In this experiment the learning algorithm is tested on non-linear transformation of sequences of spikes. The XOR problem is applied again to a network of spiking neurons, but the logic patterns are encoded by spike trains over a group of neurons, and not single spikes (see also Grünig and Sporea (2012)).

While the encoding for the XOR logic gate problem introduced by Bohte et al. (2002) requires neurons to fire a single spike, the network of spiking neurons needs a large number of sub-connections with different delays to enable the hidden and output neurons to fire at the desired times. As the problem becomes more complex such encoding might need even more sub-connections which have to be trained. The large number of weights to be trained slows down the learning process because of the large number of incoming spikes that need to be coordinated to produce the required output. This can also be seen in the previous simulations on the XOR problem where the network with 14 terminals needed almost twice as many iterations to converge as the network with 12 terminals. Moreover, it has been shown that encoding logical true and false symbols with early and late spike times respectively also requires an additional input neuron to designate the reference start time. Without the additional input neuron, even linear problems become impossible to solve (for a complete demonstration, see Section 6.2).

A more natural encoding would consist of the temporal firing patterns over groups of neurons (deCharms and Merzenich, 1996; Neuenschwander and Singer, 1996; Wehr and Laurent, 1996). In order to test such an encoding and the learning algorithm’s ability to learn non-linear patterns, the XOR problem is applied once again to a spiking neural network. In this experiment the two logical values will be encoded with spike trains over two groups of input neurons. Figure 8.3 shows
8. Simulations

Figure 8.3: (a) Network structure for the XOR problem. A feed-forward network with three layers, where the input layer consist of two groups of 20 neurons for each logical signal. (b) Each logical signal is encoded by a set of 20 spike trains, where the input signals are represented by spike trains with at least one spike, and the output signals are represented by spike trains with exactly three spikes. Reproduced after Grüning and Sporea (2012).

The network structure and the coding scheme of the two logical signals. This encoding will not necessitate multiple delays nor the additional input neuron. In all the following experiments, a single connection with no delay will be used.

8.4.1 Technical details

Each input logical value is associated with the spike trains over a group of 20 spiking neurons. In order to ensure some dissimilarity between the patterns, for each input neuron a spike train is generated by a pseudo Poisson process with a constant firing rate of \( r = 0.06/\text{ms} \) within a 30 ms time window. The minimum inter spike interval is set to 3 ms. This spike train is then split in two new spike trains by randomly distributing all the spikes (Grüning and Sporea, 2012). The
newly created spike trains will represent the patterns for the logical symbols "0" and "1". The input spike trains are required to consist of at least one spike.

The output patterns are created similarly and will be produced by one output neuron. The spike train to be split is generated by a pseudo Poisson process with a constant firing rate of $r = 0.2/\text{ms}$ within a 30 ms period of time. The resulting output patterns are chosen so that the spike trains contain exactly three spike.

Apart from the minimal network error as before, an additional stopping criterion for the learning process is introduced. The network must correctly classify all four patterns. An input pattern is considered correctly classified if the output spike train is closest to the target pattern in terms of the van Rossum distance. The network error consist of the sum of van Rossum distances between the target and actual output over the four patterns as before; a minimum value of 3 ensures that the output spikes are reproduced with an acceptable precision. Figure 8.5 shows a sample output spike train after learning along with the corresponding target pattern.

In addition to the previous experiments, an absolute refractory period is set for all neurons to $t = 3\text{ ms}$. The learning is simulated over a period of 50 ms, with a time step of 0.5 ms.

In order to determine the optimal size of the hidden layer for a higher convergence rate, different network topologies have been considered. Table 8.6 shows the convergence rate for each network topology, with a new set of spike-timing patterns being generated every trial. Figure 8.4 shows the output signals during learning for a sample trial for each of the four pattern pairs.

The learning rule is able to converge with a higher rate as the number of neurons in the hidden layer increases; a larger hidden layer means that the patterns are distributed over a wider spiking activity and easier to be classified by the output neuron. A smaller number of neurons in the hidden layer than in the input layer does not result in a high convergence rate because the input patterns are not sufficiently distributed in the hidden activity. Also, more than 100 units in the hidden layer does not result in higher convergence rates, but as the number of weights also increases the learning process is slower. Previous simulations
8. Simulations

Table 8.6: Summarised results for the non-linear classifications task.

<table>
<thead>
<tr>
<th>Hidden neurons</th>
<th>Successful trials [%]</th>
<th>Average number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>70</td>
<td>293 ± 59</td>
</tr>
<tr>
<td>60</td>
<td>54</td>
<td>301 ± 66</td>
</tr>
<tr>
<td>70</td>
<td>56</td>
<td>327 ± 91</td>
</tr>
<tr>
<td>80</td>
<td>60</td>
<td>469 ± 87</td>
</tr>
<tr>
<td>90</td>
<td>76</td>
<td>247 ± 42</td>
</tr>
<tr>
<td>100</td>
<td>76</td>
<td>439 ± 73</td>
</tr>
</tbody>
</table>

(Grüning and Sporea, 2012) show that a neural network without a hidden layer cannot learn linearly non-separable logical operations.

Figure 8.4: Output signals for all input patterns during learning for a sample trial. The x markers represent the target spike trains.

Figure 8.5 shows the input, hidden, and output signals for one of the patterns. The first 20 input spike trains represent the pattern for the logical symbol "1", while the other 20 spike trains represent the pattern for the logical symbol "0". Although the network is not responding with the exact target spike train, the output spike train is closest to the pattern representing the logical "1" than to the pattern representing logical "0" in terms of the van Rossum distance.
8. Simulations

Figure 8.5: The input, hidden, and output signals after the learning process has converged. The first 20 input spike trains represent the logical symbol "1", while the other 20 input spike trains represent the logical symbol "0". The grey signals in the output graph represent the target pattern.

8.5 Learning sequences of temporal patterns

In this experiment, we consider the learning algorithm's ability to train a spiking neural network with multiple input-target pattern pairs. The network is trained with random non-noisy spike train patterns and tested against noisy versions of the temporal patterns.

8.5.1 Technical details

The input patterns are generated by a pseudo Poisson process with a constant firing rate of \( r = 0.05/\text{ms} \) within a 100 ms period of time, where the spike trains are chosen so that they contain at least one spike. In order to ensure that a solution exists, the target patterns are generated as the output of a spiking neural network initialised with a random set of weights. The target spike trains are chosen so they contain at least two spikes and no more than four spikes. If the output patterns were random spike trains, a solution might not be representable in the weight space of the network (Legenstein et al., 2005).

The learning is considered to have converged if the network error reaches an
Figure 8.6: The output signals during learning for a sample trial. The network has been trained with 6 non-noisy patterns that span over 100 ms. The x markers represent the target spike trains.

average value of 0.5 for each pattern pair. Apart from the minimum error, the network must also correctly classify at least 90% of the pattern pairs, where the patterns are classified according the van Rossum distance. Figure 8.6 shows the output signals during learning for a sample trial. The minimum network error allows the output spike train to miss or add an extra spike, as long as the pattern is still closest to the target in terms of the van Rossum distance. The network is simulated for 120 ms with 1 ms time step.

8.5.2 The size of the hidden layer

In order to determine how the structure of the neural network influences the number of patterns that can be learnt, different architectures have been tested. In these simulations, 100 input neurons are considered in order to have a distributed firing activity for the simulated time period. The output layer contains a single neuron as in the previous simulations. The size of the hidden layer is varied from 200 to 300 neurons to determine the optimal size for storing 10 input-output pattern pairs. The results are summarised in Table 8.7. The network is able to perform better as the number of hidden neurons increases. However, a hidden layer with more than 260 neurons does not result in a higher convergence rate.
8. Simulations

Table 8.7: Summarised results for the classification task. The network is trained with 10 pattern pairs, where the size of the hidden layer is varied in order to determine the best network architecture.

<table>
<thead>
<tr>
<th>Number of hidden units</th>
<th>Successful trials [%]</th>
<th>Average number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>50</td>
<td>5 ± 0.8</td>
</tr>
<tr>
<td>210</td>
<td>52</td>
<td>6 ± 1.2</td>
</tr>
<tr>
<td>220</td>
<td>78</td>
<td>5 ± 0.6</td>
</tr>
<tr>
<td>230</td>
<td>76</td>
<td>6 ± 1.1</td>
</tr>
<tr>
<td>240</td>
<td>80</td>
<td>5 ± 0.6</td>
</tr>
<tr>
<td>250</td>
<td>74</td>
<td>7 ± 0.8</td>
</tr>
<tr>
<td>260</td>
<td>90</td>
<td>5 ± 0.7</td>
</tr>
<tr>
<td>270</td>
<td>88</td>
<td>4 ± 0.5</td>
</tr>
<tr>
<td>280</td>
<td>80</td>
<td>7 ± 2.4</td>
</tr>
<tr>
<td>290</td>
<td>90</td>
<td>4 ± 0.6</td>
</tr>
<tr>
<td>300</td>
<td>90</td>
<td>4 ± 0.4</td>
</tr>
</tbody>
</table>

8.5.3 Number of patterns

The network architectures that performed best with the lowest number of neurons (260 neurons in the hidden layer) was trained with different numbers of patterns. The results for different number of patterns are summarised in Table 8.8. The network is able to store more patterns, but the convergence rate drops as the number of patterns increases. Because the target patterns are the output spike trains of a randomly initialised spiking neural network, as the number of pattern pairs increases, the target spike trains become necessarily more similar. Hence, the network's responses to the input patterns become more similar and more easily misclassified. Since the stopping criterion requires the network to correctly classify the input patterns, the convergence rate drops as the number of pattern pairs increases.

Since the target patterns are generated as the output spike trains of a network with a set of random weights, this vector of weights can be considered the solution of the learning process. However when looking at the Euclidean distance between the weight vector solution and the weight vectors during learning, the distance
8. Simulations

Table 8.8: Summarised results for the classification task. A neural network with a hidden layer containing 260 neurons is trained with different numbers of pattern pairs.

<table>
<thead>
<tr>
<th>Number of patterns</th>
<th>Successful trials [%]</th>
<th>Average number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>100</td>
<td>7 ± 0.7</td>
</tr>
<tr>
<td>6</td>
<td>92</td>
<td>5 ± 0.6</td>
</tr>
<tr>
<td>7</td>
<td>96</td>
<td>5 ± 1.2</td>
</tr>
<tr>
<td>8</td>
<td>92</td>
<td>8 ± 1.5</td>
</tr>
<tr>
<td>9</td>
<td>88</td>
<td>7 ± 0.9</td>
</tr>
<tr>
<td>10</td>
<td>90</td>
<td>6 ± 0.6</td>
</tr>
<tr>
<td>11</td>
<td>72</td>
<td>6 ± 0.7</td>
</tr>
<tr>
<td>12</td>
<td>72</td>
<td>6 ± 0.7</td>
</tr>
<tr>
<td>13</td>
<td>58</td>
<td>5 ± 0.9</td>
</tr>
<tr>
<td>14</td>
<td>40</td>
<td>6 ± 0.9</td>
</tr>
<tr>
<td>15</td>
<td>34</td>
<td>5 ± 1.0</td>
</tr>
</tbody>
</table>

is increasing as the learning process progresses. The learning algorithm does not find the same weight vector as the solution, so multiple solutions of weight vectors to the same problem exist (for example permutations of hidden neurons is the simplest one).

8.5.4 Noise

After the learning has converged, the networks are also tested against noisy patterns. The noisy patterns are generated by moving each spike within a Gaussian distribution with mean 0 and standard deviation between 1 and 10 ms. After the network has learnt all patterns, the network is tested with a random set of 500 noisy patterns. Figure 8.7 shows the accuracy rate (the percentage of input patterns that are correctly classified) for the network with 260 spiking neurons in the hidden layer trained with 10 pattern pairs. The accuracy rates are similar for all the networks described above. The network is able to recognise more than 20% (above the random performance level of 10%) of the patterns when these are distorted with 10 ms.
8.6 Learning to Generalise

In this experiment, the learning algorithm is tested in the presence of noise. In the previous experiments where patterns were randomly generated, the learning occurred in noise free conditions. A spiking neural network is trained to recognise temporal patterns on the timescale of hundreds of milliseconds. Jitters of spike times are introduced in the temporal patterns during learning to test the network's ability to classify time varying patterns. Such experiments have been conducted with liquid state machines where readout neurons have been trained with ReSuMe to respond with associated spike trains (Ponulak and Kasiński, 2010). In this thesis, we show that such classification tasks can be achieved with feed-forward networks without the need of larger networks such as reservoirs.

8.6.1 Technical details

Three random patterns are fed into the network through 40 input spiking neurons. The hidden layer contains 210 neurons and the patterns are classified by a single output neuron. The input patterns are generated by a pseudo Poisson process with a constant firing rate of $r = 0.1/\text{ms}$ within a 500 ms time period, where the

![Figure 8.7: The accuracy on noisy patterns: The network has been trained with 10 non-noisy patterns that span over 100 ms. The random performance level is in this case 10%. The error bars show the standard error of the mean.](image)
8. Simulations

Spike trains are chosen so that they contain between 15 and 20 spikes. For the spike train generation an inter spike interval is set to 5 ms. As in the previous experiment, in order to ensure that a solution exists, the target patterns are generated as the output of a spiking neural network initialised with a random set of weights. The target spike trains are chosen so that they contain at least three spikes and no more than seven spikes. The input and target patterns are distributed over such large periods of time in order to simulate complex forms of temporal processing, such as speech recognition, that spans over hundreds of milliseconds (Mauk and Buonomano, 2004).

During learning, for each iteration, noisy versions of the input patterns are generated by moving each spike by a time interval within a Gaussian distribution with mean 0 and standard deviation varying in the range of 1 to 4 ms. Figure 8.8 shows a sample input pattern where the spikes are moved by a time interval within a Gaussian distribution with mean 0 and standard deviation 4 ms. The spikes in the target patterns are also shifted by a time interval within a Gaussian distribution with mean 0 and standard deviation 1 ms independent of the noise level in the input patterns.

A minimum average error of 0.6 for each pattern pair is required for the learning to be considered successful. During each iteration, the network is tested against a new set of 30 random noisy patterns; in order for the learning to be considered converged the network must also correctly classify at least 80% of
noisy patterns. The spike times of the testing patterns are shifted with the same distribution as the training patterns. Figure 8.9 shows the output signals during learning for a sample trial. Again, the minimum network error allows the output spike train to miss or add an extra spike, as long as the input patterns are correctly classified.

Figure 8.9: The output signals during learning for a sample trial. The network has been trained with 3 noisy patterns (4 ms jitter) that span over 500 ms. The x markers represent the target spike trains.

Figure 8.10 shows the accuracy rates on a trained network against a random set of 150 different noisy patterns, generated from the three original input patterns. The network is trained on input patterns where the spikes are moved within a Gaussian distribution with mean 0 and standard deviation 4 ms. The graph shows the accuracy rates on patterns with the spikes moved within a Gaussian distribution with mean 0 and standard deviation between 1 and 10 ms. The accuracy rates are similar to the networks trained with different input pattern jitters (1 to 4 ms). The network is able to recognise more than 50% (again above the random performance level of 33%) of the input patterns even when these are distorted with up to 10 ms.

Table 8.9 shows the convergence rate for each experiment, where the average number of iterations is calculated over the successful trials. The table also shows the number of successful trials when the network is trained on non-noisy patterns. When the network is trained with a low amount of noise in the input patterns, the learning algorithm performs slightly better than the network trained with patterns without noise. The network is able to learn even when the spike train...
Figure 8.10: The accuracy on noisy patterns: The network has been trained with 3 noisy patterns that span over 500 ms. During learning the noisy input patterns are generated by moving each spike within a Gaussian distribution with mean 0 and standard deviation 4 ms. The random performance level is in this case 33%. The error bars show the standard error of the mean.

Table 8.9: Summarised results for learning with noisy patterns. The input patterns jitter is varied between 0 and 4 ms, while the target jitter is always 1 ms.

<table>
<thead>
<tr>
<th>Input jitter during learning</th>
<th>Successful trials [%]</th>
<th>Average number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>96</td>
<td>10 ± 1.2</td>
</tr>
<tr>
<td>1</td>
<td>98</td>
<td>12 ± 1.1</td>
</tr>
<tr>
<td>2</td>
<td>95</td>
<td>19 ± 2.3</td>
</tr>
<tr>
<td>3</td>
<td>66</td>
<td>26 ± 5.6</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>115 ± 51</td>
</tr>
</tbody>
</table>

8.7 Discussion

The multilayer ReSuMe permits training spiking neural networks with hidden layers which brings additional computational power. On the one hand, the Re-
SuMe learning rule applied on a single layer (Ponulak and Kasiński, 2010) with 12 to 16 delays for each connection is not able to learn the XOR problem with the early and late timing patterns (simulations not presented here). Although the algorithm is able to change the weights in the correct direction, the network never responds with the correct output for all four input patterns. The additional hidden layer permits the network to learn the XOR problem (see section 8.2). On the other hand, a spiking neural network with the same number of units in each layer trained with SpikeProp on the XOR patterns needs at least 200 iterations to converge (see Table 8.2). Furthermore, SpikeProp requires 16 delayed sub-connections instead of just 12, hence, also implies more weights changes need to be computed. Also, SpikeProp only matches the time of the first target spike, ignoring any subsequent spikes; unlike SpikeProp, our learning algorithm also matches the exact number of output spikes.

Multilayer ReSuMe’s performance is very stable with respect to the learning parameters. The speed of learning depends mainly on the $A_+$ parameter as can be seen in Table 8.1a, where too low or too high values results in a slower convergence. Choosing a reasonable value of $A_+$, $A_-$ has a similar effect (see Table 8.1b), as too low or too high values also affect the speed of the learning process. If $A_-$ is equal or greater than $A_+$, the convergence rate starts to drop as well as the learning process is very slow.

Moreover, studies on SpikeProp show that the algorithm is unstable affecting the performance of the learning process (Fujita et al., 2008; Takase et al., 2009). Our learning algorithm is based on weight modifications that only depend on the timing of pattern pairs and not the specific neuron dynamics, therefore the search process is more stable than SpikeProp (see Figure 8.1). This can be seen in the direct comparison on the XOR benchmark. Although our algorithm also matches the exact number of spikes as well as the precise timing of the target pattern, the network learns all the patterns faster.

On the other hand, the fluctuations of the network function seen in Figure 8.1 during learning suggest that learning algorithms based on STDP processes (both supervised and unsupervised) are not very stable. Although the network error
reaches a small enough value, this is not maintained and the error soon rises, only to fall again after a few iterations. These results as well as previous similar results on the nature of STDP based learning (Grüning and Sporea, 2012) indicate that network error would maintain a minimum value only when all output patterns are identical to the target patterns. In this case, the weight modifications would be null and thus the learning would be stable. Otherwise, the error function would continually fluctuate around the solution.

The learning algorithm presented here permits using different encoding methods with temporal patterns. In section 8.3 the Iris data set is encoded using four input neurons, instead of 50 neurons required by a population encoding (Bohte et al., 2002). The simpler encoding of the Iris flower dimensions allows the network to learn the patterns in 5 times less iterations than with a population encoding used with SpikeProp (Bohte et al., 2002).

When moving from rate coded neurons to spiking neurons, an important question about the encoding of patterns arises. One encoding was proposed by Bohte et al. (2002), where logical 0 and 1 are associated with the timing of early and late spikes respectively. As the input neuron’s activity is very sparse, the spikes must be multiplied over the simulated time period, as it is known that ReSuMe performs better with more inputs (Ponulak and Kasinski, 2010). This is achieved by having multiple sub-connections for each input neuron that replicate the action potential with a different delay. The additional sub-connections, each with a different synaptic strength, require additional training. This encoding also requires an additional input neuron to set the reference start time (Sporea and Grüning, 2011). Moreover, when looking at the weights after the learning process, only some of the delayed sub-connections have a major contribution to the postsynaptic neuron while others have relatively much smaller absolute values.

The alternative to this encoding is to associate the patterns with spike trains. In order to guarantee that a set of weights exist for any random target transformation without replicating the input signals, a relatively large number of input neurons must be considered. As the input pattern is distributed over several spike trains, some of the information might be redundant and would not have
a major contribution to the output. Moreover, such an encoding does not require an additional input neuron to designate the reference start time, as the patterns are encoded in the relative timing of the spikes. The experiment in section 8.4 shows that this encoding can be successfully used for non-linear pattern transformations.

In sections 8.5 and 8.6, the target patterns are generated as the output signals of networks with random weights. Again, encodings are sparse and the corresponding pattern pairs are often locally linearly separable. As such, the transformations are more likely to be linear and as such easier to learn than in the experiments presented in section 8.2 - 8.4. The network is able to learn these transformations very fast, most of them are learnt in less than ten learning iterations. During the learning process the weights are modified in order to correctly map all input into output patterns so that they can be correctly classified as seen in Figure 8.7 and 8.10. The learning algorithm is converging fast even when the convergence rate drops significantly – when the number of hidden neuron is very low (see Table 8.7) and when the number of pattern is higher (see Table 8.8). In this case, the algorithm either converges very fast, or it does not converge within the maximum number of iterations allocated. When the algorithm is unable to converge, the weight absolute values become increasingly higher. One explanation for this behaviour the shape of the STDP learning window that maximises the weight changes as the difference between the pre- and postsynaptic spikes is very small. A more appropriate behaviour would be to have a smaller weight change if the time difference between the pre- and postsynaptic spikes is very small.

In the classification task in section 8.5, where the network is trained on 10 spike-timing pattern pairs, the learning algorithm converges with a higher rate as the hidden layer increases in size. SpikeProp can also be applied to multilayer feed-forward networks but this algorithm is limited to neurons firing a single spike (Bohte et al., 2002).

The simulations performed on classification tasks where noise was added to the spike-timing patterns show that the learning is robust to the variability of spike
8. Simulations

timing. A spiking neural network trained on non-noisy patterns can recognise more than 50% of noisy patterns if the timing of spikes is shifted with a Gaussian distribution with variance up to 4 ms (see Figure 8.7); when the network is trained on noisy patterns, it can recognise more than 50% of noisy patterns where the timing of spikes is moved within a Gaussian distribution with variance 10 ms (see Figure 8.10).

Another advantage of the learning rule is the introduction of synaptic scaling. Firstly, it solves the problem of finding the optimal range for weight initialisation. This problem is acknowledged as critical for convergence of learning (Bohte et al., 2002). Secondly, synaptic scaling maintains the firing activity of neurons in the hidden and output layer within an optimal range during the learning process. Although the firing rate of the output and hidden neurons is also adjusted by the non-correlative term \( a \) in equations (7.15) and (7.26), this is done only when the output firing rate does not match exactly the target firing rate. This can cause hidden neurons to become quiescent (neurons that do not fire any spike) during the learning process and not to contribute to the activity of the output neurons. Synaptic scaling eliminates this kind of problems by setting a minimum firing rate of one spike.
Chapter 9

Case Study: Modelling the McGurk Effect

Apart from their utilisation as artificial intelligence computing devices, it has been assumed that brain-like structures would lead to brain-like capacities and behaviour (Ellis and Humphreys, 1999). Thus cognitive modelling has emerged as an interdisciplinary study with the aim of exploring the computational properties of artificial neural networks. Although various models exist with different structures and properties, all are composed of simple computing units that are highly interconnected. In this chapter, a case study of how neural networks can be used to model a cognitive process in the human brain is described. Part of this chapter has been published in Sporea and Grüning (2010).

The current section investigates the McGurk effect by modelling it with feed-forward neural networks (Sporea and Grüning, 2010). The simulations are designed to test the two main theories about the moment when the auditory-visual integration happens. To further analyse the causes behind the McGurk illusion, the neural network that best models the effect is used to simulate the influence of language and the frequency of phonemes on auditory-visual speech perception, using two phonetic distributions from English and Japanese, with different empirical results in the McGurk effect.
9. Case Study: Modelling the McGurk Effect

9.1 The McGurk perceptual illusion

The McGurk effect is a perceptual illusion in the auditory visual speech perception domain. The effect occurs when an auditory stimulus, such as /ba/, is combined with a different visual stimulus of lips and mouth movements, such as /ga/. In the situation of an incongruent auditory-visual input, people often perceive a different sound, in this case /da/ (McGurk and MacDonald, 1976). From a phonetic point of view, the perceived sound is usually an intermediate one between the two inputs.

Studies performed on auditory-visual speech perception show the importance of facial articulators. In noisy environments, seeing the speaker’s face has a significant improvement in speech perception (Dobb, 1977; Helfer, 1997; MacLeod and Summerfield, 1990; Sumby and Pollack, 1954). The importance of visual articulation in speech perception is also emphasised by neuroimaging studies that show that during (untrained) lip-reading in the absence of auditory speech input both primary cortex (Calvert et al., 1997; Pekkola et al., 2005) and secondary cortex (Bernstein et al., 2002) are activated.

Several studies have been conducted in order to establish the moment of the auditory-visual integration during the processing of speech. While some researchers have found that the signals are processed parallel and independently and the integration occurs at a later stage (Massaro and Stork, 1998), others suggest that the integration is produced at an early point in speech processing (Bernstein, 1989; Green and Miller, 1985). Other studies suggest that the phonological repertoire influences the appearance of the McGurk effect. One such evidence is shown in Sekiyama and Tohkura (1991), where Japanese subjects have been tested for the McGurk effect. The results indicate that in noise free environments the "Japanese McGurk effect" is weaker than the English one. The perception of the incongruent auditory-visual signals, produced by a Japanese speaker, was dominated by the auditory stimuli for about 80% of the time. The most common set of incongruent syllables, auditory /ba/ combined with visual /ga/, has been heard 100% as /ba/ contrasting with the original results found by McGurk and MacDonald where for the same pair of stimuli /da/ was perceived from 64%
to 98% of the time (MacDonald and McGurk, 1978; McGurk and MacDonald, 1976). Such differences in the perception of incongruent stimuli may be caused by phonetic dissimilarities in the two languages or by cultural factors, such as the production or perception of speech.

9.2 The representation of the audio-visual patterns

The network’s input consists of patterns representing the auditory stimulus (the phoneme, which is the smallest unit of sound) and the visual stimulus (the viseme (Fisher, 1968), the basic unit of speech in the visual domain).

The phonemes are represented by a 13-element vectors which encode speech features utilised by the International Phonetic Alphabet (International Phonetic Association, 1999). The auditory patterns incorporate vectors that indicate the voice, the manner and the place of articulation. Each feature is represented by vectors; while the vectors used to encode each manner of articulation were generated randomly, the vectors used to represent the place of articulation have been constructed using the Gray code (Gray, 1953) in order to reflect their order in the vocal tract from bilabial to glottal. The Gray code is a binary system where two successive values differ by only one bit. Using the Gray code, this encoding reflects the places of articulation as they are located in the vocal tract. Table 9.1 shows the binary patterns representing the manner and place of articulation.

The visemes correspond to groups of phonemes, as the visual input contains less information than the auditory input. Therefore, several phonemes are mapped to one viseme; for example the phonemes /f/ and /v/ are in the same visual group. The visual patterns are represented by randomly generated vectors with 10 elements. The vectors are independent since viseme clusters do not have common features. Table 9.2 shows the groups of phonemes and their mapping to visemes and their encoding with binary vectors.

Table 9.3 shows the consonants in the English phonetic alphabet, the style of
9. Case Study: Modelling the McGurk Effect

Table 9.1: Binary vectors representing the main features of phonemes: (a) Manner of articulation. (b) Place of articulation.

<table>
<thead>
<tr>
<th>Manner of articulation</th>
<th>Place of articulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plosive</td>
<td>Bilabial</td>
</tr>
<tr>
<td>Nasal</td>
<td>Labiodental</td>
</tr>
<tr>
<td>Trill</td>
<td>Dental</td>
</tr>
<tr>
<td>Tap, Flap</td>
<td>Alveolar</td>
</tr>
<tr>
<td>Fricative</td>
<td>Postalveolar</td>
</tr>
<tr>
<td>Lateral fricative</td>
<td>Retroflex</td>
</tr>
<tr>
<td>Affricate</td>
<td>Palatal</td>
</tr>
<tr>
<td>Approximant</td>
<td>Velar</td>
</tr>
<tr>
<td>Lateral approximant</td>
<td>Uvular</td>
</tr>
<tr>
<td></td>
<td>Pharyngeal</td>
</tr>
<tr>
<td></td>
<td>Epiglottal</td>
</tr>
<tr>
<td></td>
<td>Glottal</td>
</tr>
</tbody>
</table>

Table 9.2: Phonemes to visemes mapping and their representation in binary vectors.

<table>
<thead>
<tr>
<th>Phonemes</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>p, b, m</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f, v</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t, d, s, z, θ, δ, t∗</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k, g, n, ŋ, j, h, N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r, w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ʃ, ʒ, tʃ, dʒ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

speech being of younger educated Americans in the Far and Mid-Western parts of the United States (International Phonetic Association, 1999). Table 9.4 shows the consonants in the Japanese phonetic alphabet, the style of speech being of educated Japanese brought up in Tokyo or areas with similar pitch accent systems (International Phonetic Association, 1999).
9. Case Study: Modelling the McGurk Effect


<table>
<thead>
<tr>
<th></th>
<th>Bilabial</th>
<th>Labiodental</th>
<th>Dental</th>
<th>Alveolar</th>
<th>Postalveolar</th>
<th>Palatal</th>
<th>Velar</th>
<th>Glottal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plosive</td>
<td>p b</td>
<td>t d</td>
<td></td>
<td></td>
<td>k g</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Affricate</td>
<td>tf d3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nasal</td>
<td>m</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fricative</td>
<td>f v θ ð s z ʃ ʒ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>h</td>
</tr>
<tr>
<td>Approximant</td>
<td>r j w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lateral Approximant</td>
<td>l</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th></th>
<th>Bilabial</th>
<th>Labiodental</th>
<th>Dental</th>
<th>Alveolar</th>
<th>Postalveolar</th>
<th>Palatal</th>
<th>Velar</th>
<th>Uvular</th>
<th>Glottal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plosive</td>
<td>p b</td>
<td>t d</td>
<td></td>
<td>k g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Affricate</td>
<td>t s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nasal</td>
<td>m</td>
<td>n</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flap</td>
<td></td>
<td>q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fricative</td>
<td>s z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>Approximant</td>
<td>j w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9.3 Rate coded neural network models

In this section the McGurk effect is modelled with two feed-forward networks of rate neurons. The two models correspond to the main theories regarding the point at which the integration of the stimuli occurs.
9.3.1 Experimental setup

The neural network has been trained with congruent audiovisual information arranged in a randomly generated sequence of a hundred patterns, replicating the way human subjects hear and see people producing sounds. The training sequence contains all the consonants from one language (e.g. English with 24 phonemes or Japanese with 16 phonemes).

Apart from the original patterns, the simulations take into consideration the influence of noise while training in order to simulate the presence of noise to audio-visual speech perception. The training sequences contain only original patterns or a random combination of original patterns and blind channel patterns (the audio or visual input has null values) and/or noisy channel patterns (the audio or visual input has each of its values inverted with a probability of 10%). The combined training sequences consist of blind channel and/or noise channel patterns with a probability of 10% depending on the type of training sequence.

The results are averaged over 100 trials, with the network being trained with a new generated sequence of patterns and a new set of random weights within the range of -0.1 and 0.1 uniformly distributed. The network is trained with the congruent pattern pairs, using the back-propagation algorithm with a learning rate of 0.1. The learning stops when the total mean squared error of the training set is sufficiently small (0.1). After each session of training, the network is tested with the sets of incongruent patterns considered to produce the McGurk effect, the winning phoneme being determined as having the smallest Euclidean distance from the original vector to the output vector.

The tables 9.6 and 9.7 below show the percentage of recognised phonemes (see 1-3 below) corresponding to the McGurk effect averaged across 100 trials, when trained with all the consonants from the English or Japanese phonetic alphabet and tested with three incongruent auditory-visual pairs of stimuli.
9. Case Study: Modelling the McGurk Effect

9.3.2 Early and late integration models

In order to test the main theories regarding the point at which the integration of the stimuli occurs, two feed-forward networks structures are used and compared. Both networks have two bands of inputs represented by binary vectors, consisting of the auditory and visual stimuli, and one set of outputs, which is the recognised sound.

The network in Figure 9.1a corresponds to the late integration hypothesis. The structure has two individual and parallel hidden layers, and a hidden integration layer. The network in Figure 9.1b corresponds to the early integration hypothesis. This structure has a hidden integration layer instead of the parallel hidden layers, without any individual pre-processing of the two stimuli. The two neural networks have the same number of neurons and have been trained and tested in the same conditions as described below.

Figure 9.1: Feed-forward network model of the McGurk effect. (a) Late integration model. (b) Early integration model. Reproduced from Sporea and Gruning (2010).
9. Case Study: Modelling the McGurk Effect

The sizes of the hidden layers in the late integration model (Figure 9.1a) have been chosen so that the network would respond with the fused patterns. The sizes of the hidden layers in the early integration model (Figure 9.1b) have been chosen to match the number of neurons in the late integration model.

Table 9.5 shows how the network responses to the incongruent pattern /b/-/g/ vary when the size of the audio hidden layer (the layer connected to the audio input layer) is constant to 7 neurons (Figure 9.1a). The size of the visual hidden layer (the layer connected to the visual input) and the size of the integration layer (the layer connected to the audio and visual hidden layers) are varied with different values. The responses to the other incongruent patterns are similar. The networks have been trained with the English phonetic alphabet, where the phonemes have equal probability of appearance or frequencies as found in conversational English (Mines et al., 1978).

While the networks trained with English phonemes with equal frequencies have similar responses divided between the auditory stimulus and the fused one, the networks trained with phonemes with English frequencies respond with a stronger McGurk effect as the visual hidden layer increases in size. Moreover, as the visual hidden layer increases its size, the network responds more with the visual pattern /g/ and less with the auditory pattern /b/.

Table 9.6 shows the summarised results corresponding to the late and early integration models in Figure 9.1. The table shows the percentage of fused responses to incongruent patterns. For all three sets of incongruent auditory-visual patterns, a significant difference between the results of the two neural network models can be seen.

A description of the notation found in the table follows:

The percentage of recognised phonemes corresponding to the McGurk effect for the incongruent sets of phonemes:

1. Audio /b/, visual /g/ - empirical data shows that is often perceived as /d/
2. Audio /p/, visual /k/ - empirical data shows that is often perceived as /t/
9. Case Study: Modelling the McGurk Effect

Table 9.5: The late integration model responses to the incongruent pattern /b/-/g/, where the audio hidden layer has 7 units, and the sizes of the visual and integration hidden layers vary. All networks have been trained with English phonemes: (a) with equal frequencies, (b) with frequencies as found in conversational English.

(a) Equal frequencies

<table>
<thead>
<tr>
<th>Visual hidden layer</th>
<th>Integration hidden</th>
<th>/b/ [%]</th>
<th>/g/ [%]</th>
<th>/d/ [%]</th>
<th>other [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10</td>
<td>57</td>
<td>0</td>
<td>38</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>59</td>
<td>0</td>
<td>33</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>15</td>
<td>40</td>
<td>0</td>
<td>48</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>18</td>
<td>54</td>
<td>3</td>
<td>38</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>20</td>
<td>41</td>
<td>0</td>
<td>45</td>
<td>17</td>
</tr>
<tr>
<td>17</td>
<td>21</td>
<td>46</td>
<td>0</td>
<td>46</td>
<td>8</td>
</tr>
</tbody>
</table>

(b) English frequencies

<table>
<thead>
<tr>
<th>Visual hidden layer</th>
<th>Integration hidden</th>
<th>/b/ [%]</th>
<th>/g/ [%]</th>
<th>/d/ [%]</th>
<th>other [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10</td>
<td>16</td>
<td>11</td>
<td>48</td>
<td>25</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>6</td>
<td>17</td>
<td>47</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>15</td>
<td>7</td>
<td>31</td>
<td>42</td>
<td>20</td>
</tr>
<tr>
<td>12</td>
<td>18</td>
<td>2</td>
<td>20</td>
<td>68</td>
<td>20</td>
</tr>
<tr>
<td>15</td>
<td>20</td>
<td>1</td>
<td>29</td>
<td>60</td>
<td>10</td>
</tr>
<tr>
<td>17</td>
<td>21</td>
<td>1</td>
<td>31</td>
<td>63</td>
<td>5</td>
</tr>
</tbody>
</table>

3. Audio /m/, visual /n/ - empirical data shows that is often perceived as /n/

The four types of random training sets used in the results tables are:

a. original patterns

b. original patterns and 10% blind channel patterns
Table 9.6: The output when trained with a random sequence of patterns having equal probability of appearance. Reproduced from Sporea and Grüning (2010).

<table>
<thead>
<tr>
<th></th>
<th>Late integration</th>
<th>Early integration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training set</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 [%]</td>
<td>2 [%]</td>
</tr>
<tr>
<td>a</td>
<td>39</td>
<td>46</td>
</tr>
<tr>
<td>b</td>
<td>27</td>
<td>23</td>
</tr>
<tr>
<td>c</td>
<td>50</td>
<td>41</td>
</tr>
<tr>
<td>d</td>
<td>32</td>
<td>45</td>
</tr>
</tbody>
</table>

c. original patterns and 10% noisy channel patterns
d. original patterns, 5% blind channel patterns, and 5% noisy channel patterns

### 9.3.3 English and Japanese phonetic alphabets

To further investigate the influence of language on the appearance of the McGurk effect, the late integration neural network is trained with English phonemes using the frequency of phonemes as found in conversational English (Mines et al., 1978) and with Japanese phonemes with frequencies found in the Japanese newspaper Asahi (Tamaoka and Makioka, 2004) and tested with the same incongruent stimuli.

Table 9.7 shows the summarised results of the simulations when the network is trained and tested in the same conditions as above. When the training set contains English consonants with English phoneme frequencies, the results show a stronger McGurk effect for all three sets of incongruent auditory-visual phonemes compared to the results of the same network trained with English phonemes with equal probabilities of appearance.

Unlike the English phonetic alphabet, the Japanese phonetic alphabet does not contain certain phonemes, such as /r/ or /l/, and contains others that do
9. Case Study: Modelling the McGurk Effect

Table 9.7: The output when trained with a random sequence of patterns having English and Japanese phoneme frequencies. Reproduced from Sporea and Grünig (2010).

<table>
<thead>
<tr>
<th>English phonemes</th>
<th>Training set</th>
<th>1 [%]</th>
<th>2 [%]</th>
<th>3 [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>68</td>
<td>83</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>68</td>
<td>80</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>42</td>
<td>53</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>48</td>
<td>75</td>
<td>72</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Japanese phonemes</th>
<th>Training set</th>
<th>1 [%]</th>
<th>2 [%]</th>
<th>3 [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>27</td>
<td>11</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>32</td>
<td>17</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>9</td>
<td>3</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>27</td>
<td>15</td>
<td>66</td>
<td></td>
</tr>
</tbody>
</table>

not exist in English, such as /N/ (International Phonetic Association, 1999). As a consequence, for the plosive sets of consonants (auditory-visual /b/-/g/ and /p/-/k/) the results are considerably lower for fusion response than the results of the equivalent model when trained with English phonemes. In the case of nasal incongruent pair (auditory-visual /m/-/n/), the results are similar when comparing to the corresponding English trained network.

In the case of Japanese phonemes, the late integration network has also been trained with patterns having equal probability of appearance and the results are similar to those of the network trained with the set of patterns with the frequency of Japanese phonemes.

9.4 Spiking neural network models

The McGurk effect is also modelled with feed-forward networks of spiking neurons trained with multilayer ReSuMe. In order to compare the spiking neuron model with the rate coded models, the same audio-visual patterns are used, where binary signals are transformed into spike timing signals using the early-late spike time encoding (see section 8.2). Thus, binary 1 and 0 are converted to input spiking
patterns as the time of a single spike at 0 and 6 ms respectively. Similarly, the output patterns are the time of a single spike at 20 and 26 ms.

9.4.1 Experimental setup

As in the case of the XOR problem, the audio-visual spike timing signals are too sparse for spiking neural networks (Bohte et al., 2002; Ponulak and Kasinski, 2010). As such, the spikes are multiplied using delays, as in Section 8.2. As in the case of the XOR benchmark, 12 sub-connections are enough as shown by preliminary simulations.

The learning is considered converged if the network error (as defined in Section 8.1.2) has reached an average value of 1.5 for each pattern pair within a maximum of 2000 iterations. This minimum network error is chosen in order to guarantee that the network has learnt to correctly classify all the patterns with an acceptable precision. As in the previous classification experiments (see Sections 8.5 and 8.6), apart from the minimum error, the network must also correctly classify at least 90% of the pattern pairs, where the patterns are classified according the van Rossum distance. The results are averaged over 50 successful trials, with the network being initialised with a new set of random weights every trial.

During each iteration, a set of 100 spiking pattern pairs randomly ordered is generated and presented to the network. After each presentation of the input pattern to the network, the weight modifications are computed for all layers and then applied. The summed network error is calculated after the presentation of each pattern pair and tested against a required minimum value.

The network parameters used in these simulations are: the threshold $\theta = 0.7$, the time constant of the spike response function $\tau = 7$ ms, the time constant of after-potential kernel $\tau_r = 12$ ms. The scaling factor is set to $f = \pm 0.005$. The learning parameters are initialised as follows: $A_+ = 1.2$, $A_- = 0.5$, $\tau_+ = \tau_- = 5$ ms, $\alpha = 0.05$. The weights were initialised with random values uniformly distributed between -0.2 and 0.8. The weights are then normalised by dividing them to the total number of sub-connections.
9.4.2 Spiking neurons model

The network topology used to model the McGurk effect is similar to the rate coded model presented in section 9.3. Figure 9.2 shows the feed-forward network used. Preliminary simulations show that one hidden layer for each input signal (auditory and visual) is sufficient for learning the audio-visual patterns.

9.4.3 English and Japanese phonetic alphabets

Table 9.8 shows the results for the spiking neural networks trained with congruent audio-visual patterns with equal frequencies and English frequencies as found in conversational English (Mines et al., 1978). Similar results are obtained with network topologies where the number of neurons in the two hidden layers vary.

Unlike the rate coded model, the spiking neural network does not respond with the McGurk illusion even when the audio-visual patterns are presented to the network with English frequencies. A slightly stronger McGurk effect can still be observed when the training patterns have the English phonemes frequencies.
9. Case Study: Modelling the McGurk Effect

Table 9.8: The output when trained with a random sequence of patterns having equal frequencies and English and Japanese phoneme frequencies.

<table>
<thead>
<tr>
<th></th>
<th>/d/ [%] in response to /b/-/g/</th>
<th>/t/ [%] in response to /p/-/k/</th>
<th>/n/ [%] in response to /m/-/n/</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal frequencies</td>
<td>8.7</td>
<td>5.8</td>
<td>11.6</td>
</tr>
<tr>
<td>English phonemes</td>
<td>18.2</td>
<td>13.6</td>
<td>25</td>
</tr>
<tr>
<td>Japanese phonemes</td>
<td>6.25</td>
<td>3.1</td>
<td>34.4</td>
</tr>
</tbody>
</table>

9.5 Discussion

The rate-coded models that simulate late and early integration have different results when tested with incongruent auditory-visual patterns. Although both rate-coded neural networks (Figure 9.1a and b) learn very well to recognise the congruent patterns, the McGurk effect in the early integration model is absent. These results support the theory of the independent parallel processing and late audiovisual integration (Massaro and Stork, 1998). When the late integration model (Figure 9.1a) is learning the auditory-visual patterns with the frequency of phonemes found in conversational English the network response to the incongruent stimuli is much closer to the experimental data (Table 9.6). In MacDonald and McGurk (1978) the percentage of fused responses to the incongruent stimuli /ba/-/ga/ is 64%, while the neural network produced the pattern /d/ 68% of times. The other sets of incongruent stimuli produce similar results to empirical data: for the /pa/-/ka/ pair McGurk and MacDonald (1976) recorded 81% fused responses and the pair /ma/-/na/ was perceived as /na/ 80% of times, while the late integration model responded with the fused pattern in 83% of times for both sets of incongruent phonemes.

The results of the simulations using the Japanese phonetic alphabet are partly consistent with empirical data showing that in noise free environments the McGurk effect is weaker for Japanese listeners (see Sekiyama and Tohkura (1991)). The results of the experiments conducted with Japanese listeners illustrate that speech
perception is almost entirely limited to the auditory stimuli when presented with incongruent signals. The results of the simulation with the late integration model (Table 9.7), although it presents a weaker McGurk effect for two sets of incongruent patterns, it also presents a strong McGurk effect for the pair /m/-/n/. These results suggest that the range of phonemes is not solely responsible for the weak "Japanese McGurk effect" found in empirical experiments. The weak "Japanese McGurk effect" may be a result of the difference in the range of consonants combined with cross-cultural dissimilarities in the perception of facial expression. Other explanations for these results can be found in the identical mapping of the viseme to phoneme used for both English and Japanese training sequences, as empirical findings show that the number of viseme clusters depends on individual speakers (Kricos, 1996).

The spiking neurons model of the audio-visual phenomenon does not respond with fused patterns when presented with the incongruent stimuli. The rate-coded McGurk effect uses the Gray code to encode the auditory patterns and thus reflects the way sounds are produced by the vocal tract. This encoding results in the appearance of the perceptual illusion in a similar way human subjects respond to these incongruent stimuli. It is well known that human subjects do not perceive fused sounds for all pairs of incongruent stimuli (MacDonald and McGurk, 1978), but only for those pairs that are produced in a certain order on the vocal tract — such as the pair formed by the auditory stimulus /b/ and the visual stimulus /g/ is perceived as /d/, which is produced somewhere in the middle on the vocal tract between the input stimuli. As this arrangement of the place of articulation is reproduced in the binary pattern, a simple feed-forward network is able to combine the two bands of inputs in such a way that when presented with an incongruent pair of inputs, the output is also somewhere in the middle — in this case the fused response.

When the binary patterns are converted to spike time patterns, the Gray encoding does not have the same meaning. The early-late coding scheme does not replicate the difference by one bit (or one piece of information) in the Gray code and thus spiking neural network is no longer able to combine the incongruent stimuli into the fused response. In this case, the early and late postsynaptic
potentials are summed together to trigger a spike at the desired time, but for incongruent pairs of input patterns, these are not enough to trigger spikes at times corresponding to the fused pattern. This is an interesting finding, as it suggests that spiking neural networks do not recognise incomplete (or in this case incongruent) patterns in the same manner rate neural networks do.

These results also suggest that, although it was demonstrated that spiking neurons are computationally more powerful (Maass, 1997b), same spatial pattern recognition and classification problems that are solved with rate coded neurons may not be solved efficiently with networks of spiking neurons.
Chapter 10

Conclusions

This thesis introduces a new algorithm for feed-forward spiking neural networks. The first supervised learning algorithm for spiking neural networks with hidden layers, SpikeProp, only considers the first spike of each neuron ignoring all subsequent spikes (Bohte et al., 2002). An extension of SpikeProp allows multiple spikes in the input and hidden layer, but not in the output layer (Booij and Nguyen, 2005). Our learning rule is, to the best of our knowledge, the first fully supervised algorithm that considers multiple spikes in all layers of the network. Although ReSuMe allows multiple spikes, the algorithm can only train single layers or readout neurons in liquid state machines (Ponulak and Kasiński, 2010).

The computational power added by the hidden layer permits the networks to learn linearly non-separable problems and complex classification and mapping tasks without using a large number of spiking neurons as liquid state machines do, or without the need of a large number of input neurons in one layer networks. Because the learning rule presented here extends the ReSuMe algorithm to multiple layers, it can in principle be applied to any neuron model, as the weight modification rules only depend on the input, output and target spike trains and does not depend on the specific dynamics of the neuron model.

The proposed learning rule has been successfully applied to classic benchmarks, such as the XOR logic gate and the Iris data set. Applying the XOR
problem, the multilayer ReSuMe can be directly compared with SpikeProp. Since multilayer ReSuMe considers all fired spikes in the output layer, unlike SpikeProp, it also matched the number of spikes. Under these circumstances, multilayer ReSuMe still requires less iterations for convergences than SpikeProp. When additional spikes are ignored in the output layer, multilayer ReSuMe is significantly faster (see Table 8.2) when tested under the same training conditions. Moreover, simulations performed on SpikeProp (see Section 6.1) suggest that SpikeProp learning process may be based on inaccurate local information.

In this thesis, a new learning algorithm for networks of spiking neurons with hidden layers is proposed and analysed. The learning rule is an extension of ReSuMe (Ponulak and Kasieński, 2010) to networks with multiple layers in the same manner that back-propagation is an extension of the delta rule. In order to compare the two generations of neuron models and their learning methods, a case study in the form of the McGurk effect model has been considered (see Chapter 9). The McGurk effect was modelled with rate-coded feed-forward networks as well as with spiking neurons. The auditory signals have been encoded using a binary system that reflects the phonemes features. While the rate-coded model responded with the fused patterns associated with this phenomenon, only a weak McGurk effect can be observed with the spiking neural network model. These results suggest that when converting binary patterns into spike time patterns, properties of the signals change. Thus, pattern recognition and classification problems with static spatial signals may not be appropriate for networks of spiking neurons.

However, temporal patterns can be explicitly represented with spiking neurons. Using spike timing patterns, we can attempt completely new problems which are computationally prohibitive to solve with classical algorithms, such as speech classification and recognition and automated lip-reading. Moreover, real data such as auditory and visual signals can be directly processed by networks of spiking neurons.

Examples of applications of spiking neural networks are automated speech recognition systems that would transform an auditory signal into a set of words
10. Conclusions

that represent responses or instructions. Studies performed on automated lip-reading systems showed that visual information can improve the performance of speech recognition especially in noisy environments (Chen and Rao, 1998; Maass and Schmitt, 1999; Wojdel and Rothkrantz, 2001). Thus, speech recognition systems can be enhanced by integrating audio and visual signals in feed-forward models such as those presented in Chapter 9.

10.1 Future work

The case study in the form of modelling the McGurk effect in Section 9 reveals certain shortcomings of modelling with spiking neurons. When converting binary signals into spike timing patterns, their properties change. Temporal patterns of audio-visual signals may produce the well-known perception illusion. Such temporal patterns can be created by encoding the phonemes and visemes features into spike times, or obtained by transforming (using signal processing filters) real signals. Using real auditory and visual signals along with biologically plausibly models of neurons we can begin to understand the mechanisms underlying the McGurk effect.

Although it can be assumed that multilayer ReSuMe works with any linearisable neuron model, the learning rule has only been tested with the Spike Response Model (Gerstner, 2001). Other neuron models, such as integrate-and-fire neurons (see Section 4.2.1), Hodgkin-Huxley neurons (see Section 4.1) or Izhikevich neurons (Izhikevich, 2007), can be considered as well.

Introducing synaptic scaling significantly improved the stability and performance of the learning process. Although simple spike timing patterns such as those in Sections 8.2 and 8.3 can still be learnt reliably without synaptic scaling, when using spike trains with multiple spike in both the input and target patterns (see Sections 8.3 - 8.5) the learning algorithm performance drops considerably. However, synaptic scaling is not the only form of homeostatic plasticity. Homeostatic intrinsic plasticity is another form of synaptic plasticity observed through experimental studies (Watt and Desai, 2010). Experimental evidence suggest that
the neuron’s intrinsic electrical properties evolve throughout life under the influence of activity dependent plasticity, including STDP (Watt and Desai, 2010). Introducing a mathematical form of intrinsic plasticity might improve the learning performance when training on non-linear complex spike train patterns (in Section 8.4 the XOR problem with spike train patterns was learnt with a convergence rate of up to 76%).

The learning rule presented here is mainly based on STDP processes. In the simulations presented in Chapter 8 the weights are updated using the exponential form of STDP (equation (7.14)). However, recent investigations into recurrent networks of spiking neurons, such as reservoir computing and liquid state machine, show that different forms of STDP result in different network dynamics (Notley and Grüning, 2012). Thus, a tri-phasic form (given by the difference between two exponential decay functions) performed significantly better in learning input-target pattern pairs in reservoir networks. An interesting extension to the current supervised learning algorithm is a version based on the tri-phasic STDP. As in the reservoir, the tri-phasic form of STDP might result in different dynamics in the feed-forward network and to an improved performance.
Appendix A. Results of SpikeProp Analysis

In this section the results of tests performed on SpikeProp are presented. The XOR benchmark was applied to a feed forward network trained with SpikeProp. The threshold has been varied between 0.2 and 1.1 and the learning rate was varied between 0.001 and 1.5; the weights have been initialised with random values in the range of 0.0 and 1.0. Table 1 shows the percentage of successful learning trials and Table 2 shows the average number of iterations for all combinations of learning rates and thresholds. Table 3 shows the standard deviation of the number of iterations needed for convergence.

Table 1: The percentage of successful learning trials of XOR problem for different combinations of learning rates and thresholds (see also Figure 6.1).

<table>
<thead>
<tr>
<th>Learning rate</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1.0</th>
<th>1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td>36</td>
<td>24</td>
<td>28</td>
<td>18</td>
<td>28</td>
<td>34</td>
<td>34</td>
<td>80</td>
<td>24</td>
<td>18</td>
</tr>
<tr>
<td>0.005</td>
<td>66</td>
<td>76</td>
<td>78</td>
<td>72</td>
<td>64</td>
<td>74</td>
<td>80</td>
<td>88</td>
<td>68</td>
<td>78</td>
</tr>
<tr>
<td>0.010</td>
<td>80</td>
<td>86</td>
<td>82</td>
<td>80</td>
<td>86</td>
<td>88</td>
<td>78</td>
<td>92</td>
<td>80</td>
<td>90</td>
</tr>
<tr>
<td>0.050</td>
<td>82</td>
<td>86</td>
<td>76</td>
<td>88</td>
<td>90</td>
<td>78</td>
<td>78</td>
<td>92</td>
<td>86</td>
<td>84</td>
</tr>
<tr>
<td>0.100</td>
<td>70</td>
<td>76</td>
<td>80</td>
<td>82</td>
<td>82</td>
<td>78</td>
<td>86</td>
<td>78</td>
<td>92</td>
<td>90</td>
</tr>
<tr>
<td>0.500</td>
<td>70</td>
<td>70</td>
<td>74</td>
<td>78</td>
<td>68</td>
<td>78</td>
<td>74</td>
<td>60</td>
<td>70</td>
<td>68</td>
</tr>
<tr>
<td>1.000</td>
<td>56</td>
<td>58</td>
<td>66</td>
<td>62</td>
<td>76</td>
<td>70</td>
<td>84</td>
<td>76</td>
<td>78</td>
<td>62</td>
</tr>
<tr>
<td>1.500</td>
<td>50</td>
<td>66</td>
<td>60</td>
<td>58</td>
<td>78</td>
<td>74</td>
<td>70</td>
<td>32</td>
<td>70</td>
<td>68</td>
</tr>
</tbody>
</table>

Table 4 shows the average number of iterations for the XOR problem when
Appendix A. Results of SpikeProp Analysis

Table 2: Average number of iterations needed for convergence for the XOR problem for different combinations of learning rates and thresholds (see also Figure 6.2a).

<table>
<thead>
<tr>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
</tr>
<tr>
<td>0.001</td>
</tr>
<tr>
<td>0.005</td>
</tr>
<tr>
<td>0.010</td>
</tr>
<tr>
<td>0.050</td>
</tr>
<tr>
<td>0.100</td>
</tr>
<tr>
<td>0.500</td>
</tr>
<tr>
<td>1.000</td>
</tr>
<tr>
<td>1.500</td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
</tr>
<tr>
<td>0.001</td>
</tr>
<tr>
<td>0.005</td>
</tr>
<tr>
<td>0.010</td>
</tr>
<tr>
<td>0.050</td>
</tr>
<tr>
<td>0.100</td>
</tr>
<tr>
<td>0.500</td>
</tr>
<tr>
<td>1.000</td>
</tr>
<tr>
<td>1.500</td>
</tr>
</tbody>
</table>

trained with SpikeProp with a momentum parameter of 0.25.

Table 5 shows the percentage of successful learning trials when the weights were initialised within the range (-0.1, 0.9) uniformly distributed, with all neurons being excitatory.
Table 3: Standard deviation of the number of iterations needed for convergence for the XOR problem for different combinations of learning rates and thresholds (see also Figure 6.3).

(a)

<table>
<thead>
<tr>
<th>Learning rate</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.2</td>
</tr>
<tr>
<td>0.001</td>
<td>886.55</td>
</tr>
<tr>
<td>0.005</td>
<td>1029.11</td>
</tr>
<tr>
<td>0.010</td>
<td>721.03</td>
</tr>
<tr>
<td>0.050</td>
<td>680.46</td>
</tr>
<tr>
<td>0.100</td>
<td>509.06</td>
</tr>
<tr>
<td>0.500</td>
<td>1223.79</td>
</tr>
<tr>
<td>1.000</td>
<td>971.98</td>
</tr>
<tr>
<td>1.500</td>
<td>1283.16</td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th>Learning rate</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.7</td>
</tr>
<tr>
<td>0.001</td>
<td>1016.52</td>
</tr>
<tr>
<td>0.005</td>
<td>1071.84</td>
</tr>
<tr>
<td>0.010</td>
<td>1022.59</td>
</tr>
<tr>
<td>0.050</td>
<td>655.32</td>
</tr>
<tr>
<td>0.100</td>
<td>545.45</td>
</tr>
<tr>
<td>0.500</td>
<td>1213.05</td>
</tr>
<tr>
<td>1.000</td>
<td>827.69</td>
</tr>
<tr>
<td>1.500</td>
<td>1166.34</td>
</tr>
</tbody>
</table>
Appendix A. Results of SpikeProp Analysis

Table 4: The average number of iterations needed for convergence for the XOR problem when trained with SpikeProp and a momentum for different combinations of learning rates and thresholds (see also Figure 6.2b).

(a)

<table>
<thead>
<tr>
<th>Learning rate</th>
<th>Threshold</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td></td>
<td>3070.21</td>
<td>2639.45</td>
<td>3481.67</td>
<td>3146.05</td>
<td>3065.56</td>
</tr>
<tr>
<td>0.005</td>
<td></td>
<td>1399.44</td>
<td>1561.7</td>
<td>1441.74</td>
<td>1348.29</td>
<td>1559.5</td>
</tr>
<tr>
<td>0.010</td>
<td></td>
<td>1067.79</td>
<td>1267.02</td>
<td>1386.37</td>
<td>1043.13</td>
<td>939.72</td>
</tr>
<tr>
<td>0.050</td>
<td></td>
<td>660.5</td>
<td>622.24</td>
<td>500.22</td>
<td>333.18</td>
<td>291.12</td>
</tr>
<tr>
<td>0.100</td>
<td></td>
<td>668.02</td>
<td>444.09</td>
<td>560.18</td>
<td>542.12</td>
<td>450.81</td>
</tr>
<tr>
<td>0.500</td>
<td></td>
<td>742.62</td>
<td>792.85</td>
<td>819.32</td>
<td>394.4</td>
<td>360.05</td>
</tr>
<tr>
<td>1.000</td>
<td></td>
<td>704.54</td>
<td>504.91</td>
<td>234.79</td>
<td>435.47</td>
<td>520.68</td>
</tr>
<tr>
<td>1.500</td>
<td></td>
<td>626.83</td>
<td>1107.14</td>
<td>759.31</td>
<td>481.41</td>
<td>907.78</td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th>Learning rate</th>
<th>Threshold</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1.0</th>
<th>1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td></td>
<td>2611.88</td>
<td>3340.1</td>
<td>3355</td>
<td>2686.56</td>
<td>3203.41</td>
</tr>
<tr>
<td>0.005</td>
<td></td>
<td>1652.12</td>
<td>1211.47</td>
<td>1448.3</td>
<td>1506.18</td>
<td>1398.43</td>
</tr>
<tr>
<td>0.010</td>
<td></td>
<td>996.88</td>
<td>1097.68</td>
<td>1242.24</td>
<td>1149.68</td>
<td>895.05</td>
</tr>
<tr>
<td>0.050</td>
<td></td>
<td>535.41</td>
<td>383.11</td>
<td>340.69</td>
<td>436.86</td>
<td>399.33</td>
</tr>
<tr>
<td>0.100</td>
<td></td>
<td>386.04</td>
<td>503.53</td>
<td>203.78</td>
<td>281.56</td>
<td>341</td>
</tr>
<tr>
<td>0.500</td>
<td></td>
<td>460.13</td>
<td>564.2</td>
<td>334.07</td>
<td>374.16</td>
<td>502.21</td>
</tr>
<tr>
<td>1.000</td>
<td></td>
<td>293.87</td>
<td>347.93</td>
<td>542.32</td>
<td>543.11</td>
<td>327.19</td>
</tr>
<tr>
<td>1.500</td>
<td></td>
<td>523.88</td>
<td>397.88</td>
<td>465.74</td>
<td>624.97</td>
<td>484.89</td>
</tr>
</tbody>
</table>
### Appendix A. Results of SpikeProp Analysis

Table 5: The percentage of successful learning trials of XOR problem with negative and positive initial weights within the range [-0.1, 0.9] (see also Figure 6.4).

<table>
<thead>
<tr>
<th>Learning rate</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1.0</th>
<th>1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td>94</td>
<td>98</td>
<td>90</td>
<td>94</td>
<td>94</td>
<td>100</td>
<td>98</td>
<td>94</td>
<td>94</td>
<td>100</td>
</tr>
<tr>
<td>0.005</td>
<td>100</td>
<td>100</td>
<td>98</td>
<td>98</td>
<td>100</td>
<td>96</td>
<td>98</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.010</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.050</td>
<td>98</td>
<td>98</td>
<td>100</td>
<td>98</td>
<td>98</td>
<td>100</td>
<td>98</td>
<td>100</td>
<td>98</td>
<td>100</td>
</tr>
<tr>
<td>0.100</td>
<td>98</td>
<td>96</td>
<td>96</td>
<td>94</td>
<td>96</td>
<td>96</td>
<td>96</td>
<td>98</td>
<td>96</td>
<td>98</td>
</tr>
<tr>
<td>0.500</td>
<td>86</td>
<td>84</td>
<td>90</td>
<td>88</td>
<td>92</td>
<td>92</td>
<td>92</td>
<td>96</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>1.000</td>
<td>60</td>
<td>62</td>
<td>60</td>
<td>68</td>
<td>66</td>
<td>82</td>
<td>72</td>
<td>84</td>
<td>88</td>
<td>80</td>
</tr>
<tr>
<td>1.500</td>
<td>48</td>
<td>64</td>
<td>64</td>
<td>74</td>
<td>66</td>
<td>70</td>
<td>72</td>
<td>84</td>
<td>78</td>
<td>82</td>
</tr>
</tbody>
</table>
Appendix B. Details of Implementation

In this section, we provide the details of the implementation of the supervised learning algorithm presented in Chapter 7. The algorithm and the simulations are implemented using the object oriented programming language Java.

The total weight change is calculated from equations 7.15 and 7.26 as a sum over all contributions of individual pre- and postsynaptic spikes pairs. In the case of weight modification for the synapse between the hidden neuron $k$ and output neuron $o$ will be:

$$\Delta w_{oh}(t) = \frac{1}{n_h} \sum_{i} \left( \sum_{j} W(t^{(f)}_h - t^{(f)}_o) - \sum_{j} W(t^{(f)}_o - t^{(f)}_o) \right)$$

where $t^{(f)}_h$ are the firing times of hidden neuron $h$, $t^{(f)}_o$ are the firing times of output neuron $o$, and $t^{(f)}_d$ are the firing times of target neuron. $W(s)$ is the learning window as defined in equation 7.14. Below is the piece of source code that calculates the weight modifications for all weights between the hidden and output neurons:

```java
dw = new double[li][lj][m]; // the weight modification matrix
for (int j=0; j<li; j++) { /* for each output neuron */
    for (int i=0; i<li; i++) { /* for each hidden neuron */
        for (int k=0; k<m; k++) { /* for each delay */
```
for (int f=0; f<s3[j].size(); f++) {  // for each output spike
    for (int l=0; l<s2[i].size(); l++) {   // for each presynaptic spike
        dw[i][j][k] += -this.learningWindow(
            s2[i].elementAt(l) + delay[k] - s3[j].elementAt(f));
    }
}
}

for (int j=0; j<lj; j++) {  // for each target output neuron
    for (int i=0; i<li; i++) {   // for each hidden neuron
        for (int k=0; k<m; k++) {   // for each delay
            for (int f=0; f<td[j].size(); f++) {   // for each target output spike
                for (int l=0; l<s2[i].size(); l++) {   // for each hidden spike
                    dw[i][j][k] += this.learningWindow(
                        s2[i].elementAt(l) + delay[k] - td[j].elementAt(f));
                }
            }
        }
    }
}

The weight modifications for the synapses between the input and hidden neurons are calculated in a similar manner.
References


Gray, F. Pulse code communication, March 1953.


REFERENCES


REFERENCES


