A thesis in partial fulfillment of
the requirements for the degree of
Master of Science in Artificial Intelligence

Delay learning in a biologically plausible reservoir computing model

Arvind H.R. Datadien
adatadien@gmail.com

Supervised by
Dr. I.G. Sprinkhuizen-Kuyper
Dr. M.A.J. van Gerven

August 15, 2013

Department of Artificial Intelligence
Radboud University Nijmegen
The Netherlands

Abstract

In this thesis, a reservoir model created by Paugam-Moisy et al. (2008) was replicated, and subsequently modified to be more biologically plausible. The use of a different neuron model and STDP function had no negative impact on performance. For the delay learning mechanism, replacing delay adaptation with delay selection proved viable, but did result in somewhat lower performance. It was discovered here that the reservoir in this model served no purpose. Therefore, the use of a small-world Watts-Strogatz reservoir instead of a randomly connected reservoir, could not affect performance.

Acknowledgements

I would like to thank Ida Sprinkhuizen-Kuyper and Marcel van Gerven for their guidance and feedback during this project. Also a big thank you to Louis Vuurpijl for agreeing to be the third reader.

Contents

Abstract **i**

1 Introduction **1**
1.1 Research questions . . . 3

2 Methods **6**
2.1 Neuron model . . . 6
2.2 Network structure . . . 8
2.3 Reservoir learning . . . 9
2.4 Output learning . . . 11
2.4.1 Delay adaptation . . . 11
2.4.2 Delay selection . . . 12
2.5 Input patterns . . . 13

3 Experiments **16**
3.1 Comparing delay learning methods . . . 16
3.2 Comparing reservoirs . . . 17

4 Results **19**

5 Discussion **24**
5.1 Delay learning methods . . . 24
5.2 Reservoirs with full input . . . 25
5.3 Reservoirs with partial input . . . 27

6 Conclusion **29**
6.1 Future work . . . 30

A Appendix **32**
A.1 Software implementation . . . 32
A.2 Attempts at supervised delay selection . . . 33

Bibliography **34**

Chapter 1

Introduction

Networks of spiking neurons encode information via the timing of individual spikes. This contrasts with more traditional neural networks in which neurons communicate through average firing rates. Spiking neural networks (SNNs) have two very attractive qualities: their computational power, and similarity to the biological brain [13]. However, one important open question regarding these networks, is how to train them effectively. Interestingly though, it seems that this question does not always need an answer.

It was shown that an untrained SNN can be very effective when used as the liquid in a liquid state machine (LSM). An LSM is a computational model that consists of a filter (called the "liquid") that transforms input signals into an internal state, and a readout function that maps this internal state to an output value [14]. One advantage of this model is that training is only required for the readout function, not for the entire liquid.

The liquid is often implemented by a randomly connected SNN, which can be said to perform mainly two functions. The first is temporal integration. If the network contains recurrent connections, it can exhibit short term memory, meaning that past inputs can influence the current state of the liquid. This can be beneficial for tasks such as speech recognition, where data over time is needed to recognise spoken words. The second, and perhaps most important function of the liquid is to transform an input signal non-linearly into a high-dimensional space, similar to what is done with support vector machines. Once in this space, inputs can be easier to classify and may be linearly separable [20].

The readout function can be implemented in many ways. In an influential paper by Maass (2002) [14], a real-valued output was attained by connecting each liquid neuron to multiple spiking output neurons. Within a time window of 20 ms, the proportion of output neurons that fired was interpreted as the output value. It was stated that for a binary value, a single output neuron could be used, interpreting its firing within the time window as a true value, and a non-firing as false. In the paper by Maass, the weights of the connections between liquid and output neurons were changed with a modified version of the delta rule (the p-Delta learning rule for parallel perceptrons [1]), to attain the desired outputs, given certain inputs.

Simultaneously with the development of the LSM by Maass, the very similar Echo State Network (ESN) was conceived by Jaeger [11]. This model used an untrained, recurrent network of traditional (sigmoid) neurons, to create non-linear transformations of input values. These internal states were sent to output neurons with weights that could be modified through linear regression to obtain the desired output values.

The approach used in LSMs and ESNs is now often referred to as Reservoir Computing, where "reservoir" refers to the presence of a liquid or recurrent neural network. Models in this field have been successfully used for speech and handwriting recognition, robot control, financial forecasting, brain computer interfacing, and many other tasks [12] [17]. Originally, one of the benefits of reservoir computing was that only the output weights needed to be trained, and not the recurrent network. More recently, however, researchers also began training the reservoir, with the goal of creating one that performs optimally for a specific task. Still, such reservoir computing models can be distinguished from most other neural network models, by their use of a different training method for the reservoir and for the output [12].

One such reservoir computing model where the reservoir was trained differently than the output, was proposed by Paugam-Moisy et al. (2008) [16]. This model was the basis for our research, and may also be referred to in this thesis as the "PM model" or the "original model". It consisted of spiking neurons, arranged in an input layer, a randomly connected reservoir, and an output layer, as can be seen in Figure 2.2.

Training in the reservoir was done through Spike-Timing-Dependent Plasticity (STDP). This biologically observed, unsupervised learning method changes the weight of a connection based on the time interval between a presynaptic spike arrival and a postsynaptic firing. In the PM model (as in most computational models with STDP) the weights of excitatory connections were increased for positive intervals (if a spike arrived before a neuron fired), and decreased for negative intervals. The size of these weight changes can be determined with a function such as the one displayed in Figure 2.4, however, many other forms of STDP also exist [3]. An in-depth, and very recent review of STDP in biological neurons is given in [5].

One concept that is often overlooked in SNNs, but was present in the PM model by Paugam-Moisy et al., is that of delays. By this we mean the time between the moment a presynaptic neuron fires, and the moment that a postsynaptic neuron's membrane potential is influenced by the event. In the brain, this delay could be caused by the time it takes for signals to travel along axons, synapses, and dendrites. Axonal transmission delays alone were found to vary between 0.1 and 44 ms, and to be reproducible with sub-millisecond precision [9].

Because the reservoir in the PM model incorporated STDP and delays, it could contain Polychronous Groups (PGs). This concept was first described by Izhikevich (2006) [9]. There, a polychronous group was defined as a collection of neurons that fire together not synchronously, but in a reproducible time-locked pattern that is based on the connectivity (delays and weights) between neurons. PGs can be created when an input pattern is presented repeatedly to a network. STDP then strengthens certain connections, forming new PGs that become active when the input pattern is seen again. In this way, polychronous groups can be seen as memories of past inputs. Interestingly, because a neuron can take part in multiple groups, a network can potentially contain many more PGs than neurons, resulting in a large memory capacity for such networks [9].

In their model, Paugam-Moisy et al. observed that some PGs became active selectively for a class of inputs. Therefore, if the activation of these groups could be detected, input could be classified. To do this, output neurons were used that received connections from reservoir neurons, with delays that could be modified. By setting these delays to a configuration where spikes from a class-specific PG would all arrive simultaneously at an output neuron, that neuron could fire selectively for that PG, and therefore detect the class of an input pattern. The right delays were found with a novel supervised delay learning algorithm. Using these methods, the model was reported to be quite successful at classifying images of handwritten digits [16].

1.1 Research questions

Many parts of the PM model by Paugam-Moisy et al. are biologically plausible. For example, the spiking neuron is based on the biological neuron, and STDP and delays have been observed in the brain. There are however some parts that were not based on biological findings. In this thesis, we intended to explore if the model could be made more biologically plausible by replacing some of these parts with more biologically plausible alternatives. If so, its functioning may be similar to (part of) the brain, and thus teach us about this complex system.

The first aspect of the model employed by Paugam-Moisy et al. that is not very biologically plausible, is the delay learning method that was used to train the output neurons.

Here we will refer to this concept of directly altering delays, as delay adaptation, although it is also known as delay shift [4]. Theoretically, delay adaptation could occur in the brain by modifying the thickness of axons or dendrites, the amount of myelination, or properties of synapses, but to our knowledge, no biological learning mechanisms have (yet) been found that work like this.

However, another form of delay learning exists that does not require the modification of delays. It is called delay selection, and makes the assumption that multiple pathways between neurons exist, with different (static) delays. Pathways with desirable delays can then be selected by increasing their weights, and decreasing the weights of others.

The concept of delay selection was proposed in a paper by Hopfield (1995) [7]. There it was suggested that by using delay selection, a neuron can learn to respond maximally to a spatiotemporal spike pattern. If this neuron, after learning, receives an input pattern that matches its delay configuration, all input spikes will arrive simultaneously, and the neuron will fire early. If the input pattern only partially matches its delays, the neuron will fire later, or not at all. By interpreting an early firing as a high value, and a later firing as a lower value, the neuron can be said to act similarly to a radial basis function unit [7].

This idea was implemented in a model by Natschlager and Ruf (1998) [15] where it was used to cluster input data. The model was expanded upon by Bohte et al. (2002) [2], who used a different encoding mechanism and a network with multiple layers to detect more difficult to find clusters. In both these models, weights were modified by an artificial learning rule, which was later shown to be replaceable with a biologically observed STDP function [18], making the concept of delay selection even more biologically plausible.

Therefore, the first aspect of the PM model that was changed here, was the delay learning mechanism. We replaced the artificial delay adaptation method with the more biologically plausible delay selection method that uses STDP.

The second part of the model that was not entirely biologically plausible was the randomly connected topology of the reservoir. It has been suggested [6] that the brain may be organised like a small-world network, containing many local, but also some long range connections. Together with the fact that they may be more biologically plausible, small-world networks were recently shown by Vertes and Duke (2010) [21] to be capable of containing many more polychronous groups than randomly connected networks. Since the original PM model was stated to work by detecting activated PGs, we hypothesised that with the presence of more polychronous groups, performance would improve.

For these reasons, the second aspect of the model that was modified, was the reservoir. Specifically, we used a Watts-Strogatz [22] network with small-world characteristics, instead of a randomly connected network.

Replacing the delay learning mechanism, and the reservoir topology with more biologically plausible alternatives, would result in a model in which most aspects are based on biological findings. Therefore, analysis of this model may provide new insights on how the brain works. The modifications could also lead to improved performance, because the biological brain has evolved over time to be able to perform certain tasks very effectively. Therefore, our research questions for this thesis were as follows:

Can the model by Paugam-Moisy et al. be made more biologically plausible by training the output neurons with delay selection, and using a small-world network architecture for the reservoir? Will these modifications lead to performance gains compared to results obtained with the original model?

In the following chapters we will first describe the details of the model that was used. Following that, the experiments that were conducted with this model will be explained. The results of these experiments will then be presented, and subsequently discussed. Finally, we present our conclusions, and suggestions for future work. An appendix is also included, containing extra information.

Chapter 2

Methods

Our experiments were conducted with a custom-built software application, able to simulate a spiking neural network. In this section we will discuss the workings of this network. More low-level details of the software implementation can be found in the appendix.

In the following sections, we will first present the building block of our network: the spiking neuron. Then the structure of the network will be explained, including the input, reservoir, and output layer. Following that, the training methods for the reservoir, and the output layer will be discussed separately. Finally we will discuss what input is given to the network.

2.1 Neuron model

All neurons in the network were simulated with Izhikevich's simple model of spiking neurons [8]. This model can function similarly to the biologically plausible Hodgkin-Huxley model, even though it is much more computationally efficient. In the article by Paugam-Moisy et al. [16], the zeroth order spike response model (SRM$_0$) was used, but here we chose to replace this with the Izhikevich model, in order to further increase the biological plausibility of our network.

The Izhikevich model uses a variable v to represent a neuron's membrane potential, and a variable u which is called the membrane recovery variable, and gives negative feedback to v. Two differential equations v', and u' are used to model the changes of these variables over time, and are specified as follows:

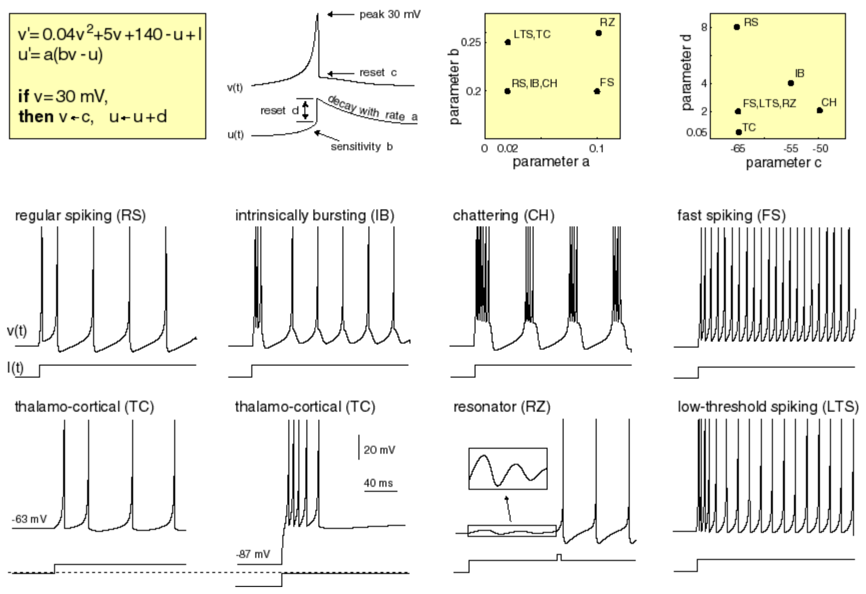


FIGURE 2.1: Izhikevich simple model of spiking neurons. By choosing different parameter values (shown in the two top right graphs), neurons with different spiking behaviours can be modelled. Electronic version of the figure and reproduction permissions are freely available at www.izhikevich.com.

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{2.1a}$$

$$u' = a(bv - u) \tag{2.1b}$$

Here, I represents the input received through synapses from other neurons. For spikes from excitatory neurons, I will be increased, while the reverse is true for spikes from inhibitory neurons. If multiple excitatory spikes arrive within a small amount of time, and the membrane potential reaches 30 mV, the neuron is said to have fired, and v and u are reset according to the following equation:

$$\text{If } v \geq 30 \text{ mV, then } v \leftarrow c, \text{ and } u \leftarrow u + d \tag{2.2}$$

By choosing different values for parameters a, b, c, and d, different types of neurons can be modelled. These neurons and their behaviours are discussed in detail in [8], but an overview is given here in Figure 2.1. For our application we chose parameter values so that excitatory neurons would behave as regular spiking (RS), and inhibitory ones would act as fast spiking (FS) neurons. The values for RS neurons are: $a = 0.02, b = 0.2, c = -65, d = 8$. The values for FS neurons are: $a = 0.1, b = 0.2, c = -65, d = 2$.

Time in the network is modelled with discrete time steps of 1 ms. Each time step, membrane potentials are calculated, firing neurons are detected, travelling spikes are

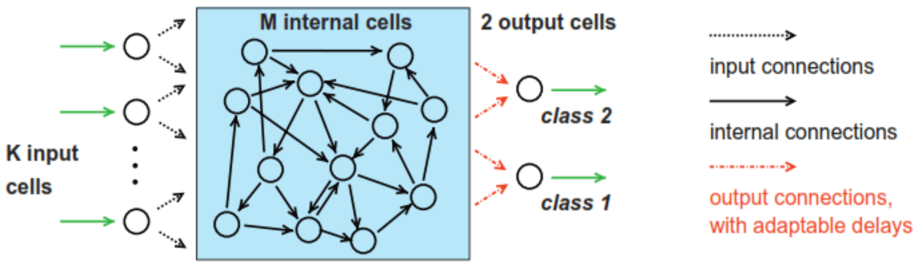


FIGURE 2.2: Overview of the reservoir model by Paugam-Moisy et al. Figure taken from [16]. Input neurons are connected to the reservoir. Reservoir neurons are connected to each other, and to the output neurons. In this model, delay adaptation was used, so delays of connections to the output neurons can be modified.

advanced, and weights are modified. A smaller time step could increase realism, and in some cases performance (as was also reported in [16]). It would, however, come at the cost of much longer computation times.

Connections between neurons have a weight, and a delay. The weight indicates the influence a presynaptic spike has on a postsynaptic neuron's membrane potential. The delay is the amount of time that passes between the moment a presynaptic neuron fires, and the moment that the spike arrives at a postsynaptic neuron.

2.2 Network structure

The network topology is based on the reservoir model presented in [16]. It consists of an input layer, a reservoir, and an output layer, with each part connected to the next, as is illustrated in Figure 2.2. The function and structure of each layer will be discussed in the following paragraphs.

The input layer's function is to receive input patterns from an external source and to pass them onto the reservoir. To achieve this, each connection between the input and reservoir has a delay of 1 ms, and a weight of 20, which is enough to induce an immediate firing in the receiving reservoir neuron. Input neurons are connected randomly to excitatory reservoir neurons with a probability P_{in}, which varies between experiments.

The reservoir's function is to convert input patterns into a form of activity that should be easier to classify by the output neurons than the input itself. Interestingly, some reservoirs should be able to exhibit more useful activity than others. To be able to test this, our model supports three different reservoirs. All three consist of 80 excitatory and 20 inhibitory neurons, but differ from each other in the way that their neurons are connected. Here they will be called the unconnected, the random, and the Watts-Strogatz (WS) reservoirs, and they will be described in the following paragraphs.

Neurons in the unconnected reservoir receive spikes from input neurons, and send them to output neurons, but have no connections with other reservoir neurons. The unconnected reservoir is therefore not so much a reservoir, but more a collection of neurons that forward input firings unchanged to the output neurons.

In the random reservoir, neurons are connected randomly to each other with a probability of 0.3, with the exception that there are no connections between inhibitory neurons. Each connection is initialised with a random, static delay between 1 and 20 ms. Excitatory connections start with a random, plastic weight between 0 and 10, and inhibitory connections start with a static weight of -5. These settings are largely based on those reported in [9], in order to allow the formation of polychronous groups.

The topology of the Watts-Strogatz reservoir is based on the WS network described in [21], and is illustrated in Figure 2.3 for an excitatory and an inhibitory neuron. It consists of an outer circle with excitatory neurons, and an inner circle with inhibitory neurons. Excitatory neurons are connected to the 24 nearest excitatory, and the 6 nearest (opposing) inhibitory neurons. Inhibitory neurons are only connected to the 24 nearest (opposing) excitatory neurons. To add some level of randomness, connections are then selected with a probability of 0.3, and modified to connect to another randomly selected neuron, as was also done in the original Watts-Strogatz model [22]. The weights and delays of connections here, are initialised in the same way as in the random reservoir.

The last layer in the network is the output layer. The function of this layer is to learn to classify an input pattern based on its resulting reservoir activity. Two output neurons were used here; one for each input class. The way these neurons are connected is different for the two delay learning methods, and will therefore be discussed in Section 2.4.

2.3 Reservoir learning

The reservoir is trained with Spike-Timing-Dependent Plasticity (STDP). This unsupervised learning rule modifies weights on the basis of spike timings. The amount by which a weight is changed, depends on the interval between the last postsynaptic firing time, and the last presynaptic spike arrival time ($t = t_{firing} - t_{spike}$). The STDP function that was used in [16] was more abstract, but here we use a function that is based on the behaviour of biological neurons. It is illustrated in Figure 2.4, and defined as follows:

$$\Delta w(t) = r \cdot \begin{cases} A_{-} \cdot e^{-t/\tau_{-}} & \text{if } t < 0 \\ A_{+} \cdot e^{t/\tau_{+}} & \text{if } t \geq 0 \end{cases} \tag{2.3}$$

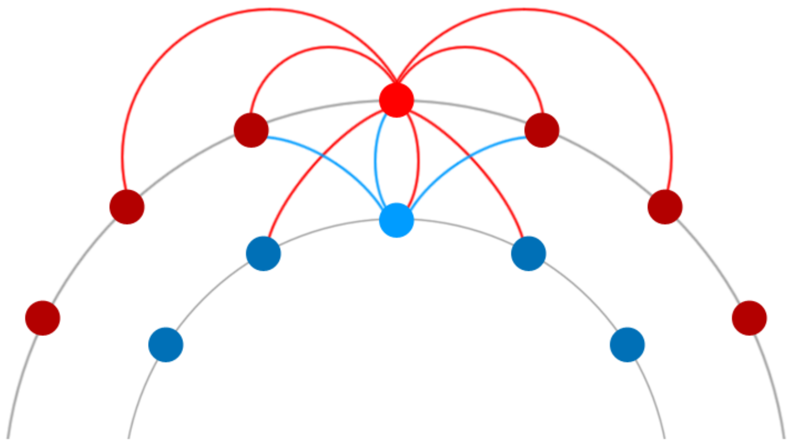


Figure 2.3: Internal connectivity of the Watts-Strogatz (WS) reservoir. Here, excitatory neurons are colored red, and placed on the outer circle, while inhibitory neurons are blue and located on the inner circle. For clarity, only a few of the connections that emanate from the two middlemost neurons are shown here. In the actual WS reservoir, each excitatory neuron is connected to the 24 nearest excitatory, and 6 nearest inhibitory neurons. Each inhibitory neuron is connected only to the 24 nearest excitatory neurons. Some of the connections are then modified to connect to a randomly selected neuron. This randomisation is not displayed here.

The weight changes that result from this function are applied immediately after each spike arrival, and after each firing. When a spike arrives at a synapse, the postsynaptic neuron has fired some time in the past. Therefore, because $t < 0$, the weight will be decreased to some degree. When a neuron fires, that neuron has received spikes in the past. This means that $t \geq 0$ for all its receiving connections, and that their weights will be somewhat strengthened.

Our chosen parameter values for STDP were based on [9], where it was shown that they resulted in the formation of polychronous groups. The only difference is that we used a slightly lower value for A_-, because this seemed to result in somewhat better performance. Unless stated otherwise, the following values were used: $A_+ = 1.0$, $A_- = -1.4$, $\tau_+ = 20$ ms, $\tau_- = 20$ ms, $r = 0.05$.

Most other aspects of our STDP implementation were also taken from [9]. This means that STDP was only applied to excitatory connections; inhibitory connections were static. Also, weights were cut off to stay between the minimum and maximum values of 0 and 10 for excitatory connections. Finally, connections were slightly potentiated when spikes arrived at neurons that (almost) never fired. This was done in order to make silent neurons active again. More concretely, whenever a spike arrived at a neuron whose last

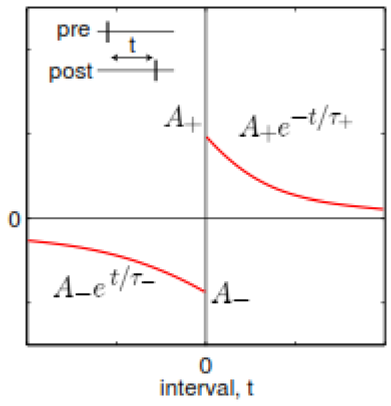


FIGURE 2.4: A graph of the STDP function. This image was taken from [9]. Here, t is the interval between a presynaptic spike arrival, and a postsynaptic firing for a synapse. The vertical axis displays the amount by which the weight of the synapse will be changed.

firing time was more than 100 ms ago, the weight of the connection was increased by $0.1 \cdot r$ (where r is the same learning rate parameter as in Equation 2.3).

2.4 Output learning

Connections from the reservoir to the output neurons are modified with a delay learning mechanism. Our model supports two forms of delay learning: delay adaptation, and delay selection. These will be discussed separately in the following sections.

2.4.1 Delay adaptation

Delay adaptation works through the direct modification of delays. In our implementation, one connection exists between each excitatory reservoir neuron, and each output neuron. These connections are initialised with randomly selected delays between 1 and 20 ms, and weights of 5. This weight value was chosen so that, similarly to the original model, four simultaneously arriving spikes are able to make an output neuron fire. After initialisation, the weights of these connections never change; they are static and can not be modified by STDP or any other weight-changing mechanism. Their delays however, *can* be changed. This is done as described in Algorithm 1, which was taken from [16] and will be further explained below.

During simulation, input patterns are presented that cause reservoir activity, which results in the firing of output neurons. For each input pattern, we want the output neuron representing the class of that pattern to fire first, and the other output neuron to fire at least $\mu = 5$ ms later. If this is not the case, delays should be adjusted in order

Algorithm 1 Delay adaptation algorithm

```
for all patterns do
    present pattern to the network
    define target output neuron based on the current pattern's class
    if target fires less than $\mu$ ms before non-target, or after non-target then
        select triggering connection of target neuron, and reduce delay by 1 ms
        select triggering connection of non-target neuron, and increment delay by 1 ms
        trim delays of both connections to stay between $d_{min}$ and $d_{max}$
    end if
end for
```

to achieve the desired result for the next pattern presentation. Delays are adjusted by decreasing the delay of a triggering connection for the neuron that should fire first, and increasing such a delay for the neuron that should fire last. A triggering connection here, is defined as a connection on which an output neuron received its last spike, before firing. If there are more than one triggering connections, because multiple spikes arrived simultaneously, a random connection is selected from them. Additionally, delays are always kept between the minimum and maximum values of 1 and 20 ms.

In the original PM model, the output neurons were restricted from firing more than once during a pattern presentation. In our model however, output neurons can fire whenever they want, but only the first firing of each output neuron during a pattern presentation is taken into account.

2.4.2 Delay selection

With delay selection, multiple connections exist between each reservoir and output neuron. Although the delays of these connections are static, the weights are not. Note that this is the opposite of the setup for delay adaptation, where weights are static, and delays are not. With delay selection, weights of connections are modified by STDP, so that connections with proper delays can be potentiated (selected), while others become unused.

In our implementation, each excitatory reservoir neuron has 20 connections to each output neuron. These connections have delays with incremental values, meaning that there is a 1 ms delay for the first, 2 ms for the second, and 20 ms for the last connection. Again, these delays are static, they are never modified. The weights of the connections are initialised to random values between the minimum of 0, and the maximum of 0.2. A lower maximum weight was used here than in the reservoir, because there are multiple connections between neurons, and therefore multiple spikes are sent to the output neurons for each reservoir firing. To modify these weights, the same STDP function was used as in the reservoir (see Equation 2.3), but with a lower learning rate of $r = 0.002$, due to the lower maximum weight.

Besides connections from the reservoir, the output neurons also receive inhibitory connections from each other. These connections have a 0 ms delay, and static weights strong enough to prevent the output neurons from firing simultaneously. By prohibiting simultaneous firings, neurons are prevented from learning to detect the same pattern. The same mechanism (called lateral inhibition) was also used in the delay selection implementation by Natschlager et al. [15]. It is required for delay selection because, due to its unsupervised nature, there is otherwise no way to influence which neuron will learn to detect which pattern. It is not used with delay adaptation, because the supervised learning algorithm there already teaches neurons to detect different patterns.

When a simulation is run with delay selection, initially output neurons will fire randomly in response to input patterns. However, the desired response is for each output neuron to fire first for only one class of input patterns. This can be achieved by STDP, because it potentiates connections that repeatedly cause a neuron to fire, and depresses others. This results in a neuron that fires only in response to a specific spatio-temporal spike pattern. This neuron is then able to detect input patterns that are similar to this spike pattern, which ideally would be all input patterns that belong to one class.

One major difference between delay selection and delay adaptation, is that with the former, we have no control over which output neuron learns to detect which class of inputs. This is due to the unsupervised nature of STDP with which delay selection works. It means that, while delay selection is able to find clusters of similar inputs, delay adaptation can also associate (label) these clusters with a class. To still be able to compare results between the two learning methods, clusters that are found with delay selection are labelled manually. This is done by simply labelling each of the two clusters with a different class in the way that leads to the highest success rate for a simulation.

2.5 Input patterns

As mentioned before, interaction with the network is done by feeding it input patterns, and observing the resulting activity of its output neurons. These input patterns are simply the times at which input neurons are made to fire. After the firing of input neurons, there is time for reservoir and output neurons to fire as a result of this, and for neurons to return back to their resting state before the next pattern is presented. To be able to test the network with tasks of varying difficulty, the same input patterns

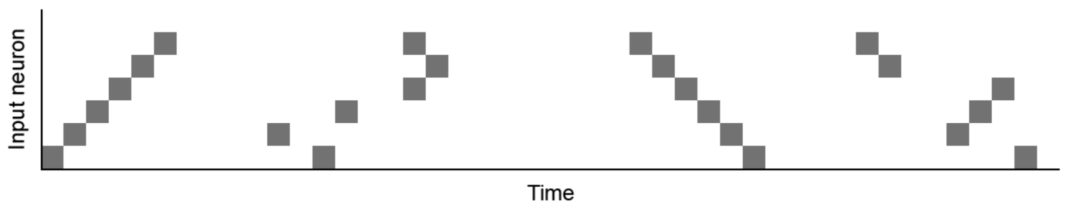


FIGURE 2.5: Abstract input patterns. The two left patterns belong to the first class, and the two right patterns belong to the other. For each class, a pattern is shown with, and without noise. It can be seen that patterns without noise are easier to distinguish from each other. Although here, only 6 input neurons are shown, the abstract patterns that were used consisted of firing times for 10 input neurons.

are used as in [16]. Here we call them abstract, and digit patterns, and they will be presented in the following paragraphs.

The abstract patterns have no underlying meaning, they are simply the firing times of 10 input neurons, as chosen by us. We used two classes of abstract patterns, which differ from each other by their firing times, not by the neurons that are fired. Both classes can be seen in Figure 2.5. To make patterns more difficult to cluster, noise can be added. This is done for each pattern by shifting each firing back or forward in time by an amount between 0 and n ms, chosen randomly from a uniform probability distribution.

The digit patterns are based on grayscale images of handwritten digits from the USPS data set. Examples are shown in Figure 2.6. For these patterns, we used 256 input neurons, one for each pixel of the 16x16 images. The grayscale of each pixel was converted linearly into the firing time of its associated input neuron, with a firing time of 0 ms for black, and 20 ms for white pixels. The dataset consisted of a separate training and testing set for each digit. The training sets contained on average 687 patterns per digit, while the testing sets made do with 192.

As described in the previous section, the network's response consists of the firing times of its output neurons, with each output neuron representing a class. During and shortly after the presentation of an input pattern, the first output neuron that fires is interpreted as the network's response. That neuron signifies the class to which the input pattern belongs, according to the network. Using this response, and the aforementioned input patterns, the performance of the network was measured during multiple experiments. The setup of these experiments will be discussed in the next chapter.

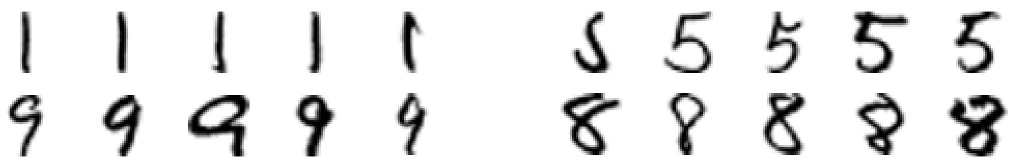


FIGURE 2.6: Selection of handwritten digits from the USPS dataset. Images in the dataset consist of 16 by 16 grayscale pixels. These are converted into input patterns that consist of firing times for 256 input neurons. Here, samples of five images are shown for digits 1, 9, 5, and 8. The complete training sets contain 1005, 644, 556, and 542 images for each digit, respectively. The testing sets contain 264, 177, 160, and 166 images, respectively.

Chapter 3

Experiments

The previously described model was used to conduct two experiments that would help answer our research questions. In the first experiment, the delay learning mechanism was varied, while in the second experiment, different reservoirs were used. The setup for both experiments will be presented here.

3.1 Comparing delay learning methods

The first experiment was conducted to compare the performance of delay adaptation to that of delay selection. If delay selection performed adequately, it could be seen as a biologically plausible alternative to delay adaptation.

The setup here was very similar to that of [16]. In all cases, a randomly connected reservoir was used. For delay adaptation, one connection existed between each reservoir and output neuron, with a static weight, but plastic delay. For delay selection, these were replaced by 20 connections with plastic weights, but static delays.

Performance was tested on five tasks of varying difficulty. For each task, twenty simulations were run, and the results were averaged. Each simulation consisted of three phases: initialisation, training, and testing. During the initialisation phase, reservoir learning through STDP was active, but output learning was disabled. This phase was used to balance activity in the reservoir, before output training began. It was required for some setups (with high weights, many connections, or few inhibitory neurons) where the reservoir would initially exhibit excessive, non-stop firing behaviour. The training phase was used for actual learning; both STDP in the reservoir and delay learning at the output were active. The testing phase was used to measure the network's performance. Therefore, all learning methods were disabled during this phase.

The first three tasks consisted of the classification of abstract patterns with 0, 4, or 8 ms noise. These patterns were provided through 10 input neurons, each connected randomly to each reservoir neuron with probability $P_{in} = 0.125$. The patterns of both classes were presented alternately, 400 of them during initialisation, 2000 during training, and 400 during testing.

The last two tasks concerned the classification of digits 1 and 9 (easy), and digits 5 and 8 (difficult). These were provided through 256 input neurons, each connected randomly to each reservoir neuron with probability $P_{in} = 0.0125$. The patterns for each digit were separated in a training and a testing set, the latter being used only for the testing phase. These sets were trimmed to contain an equal number of patterns for each digit that was used. The presentation of all available (training, or testing) patterns for both digits in random order, will be called here an epoch. Training epochs contained around 1100 to 2000 patterns, and testing epochs 300 to 500, depending on the digits that were used. The initialisation phase lasted for 1/5th of an epoch, the training phase for 8 epochs, and the testing phase for 2 epochs.

3.2 Comparing reservoirs

The second experiment was conducted to compare the performance of a small-world Watts-Strogatz reservoir to that of a random reservoir. Because the small-world reservoir was shown to have a larger capacity for polychronous groups, it was hypothesised to lead to improved performance. To be thorough, we also tested the performance of an unconnected reservoir. Because this “reservoir” contains no internal connections, it can contain no polychronous groups at all.

Three tasks were conducted to test the performance of these reservoirs. The tasks were learnt with the original delay adaptation learning mechanism. Again, performance for each task was measured during twenty simulations, and the results were averaged. The simulations were run in the same way as in the first experiment.

The easier variants of tasks were skipped in this experiment, in order to save time. Therefore, the first task consisted of the classification of digits 5 and 8. Again, 256 input neurons were used, and connected to reservoir neurons with $P_{in} = 0.0125$.

In the second task, abstract patterns with 8 ms noise were to be classified. Here 10 input neurons were connected to the reservoir the same as before, with $P_{in} = 0.125$. Because most of the reservoir neurons received a connection from an input neuron, this setup will be referred to as “full input”.

In the last task, abstract patterns with 8 ms noise were to be classified again. This time, however, fewer connections existed between the 10 input neurons and the reservoir, due to the use of parameter value $P_{in} = 0.0125$. This task was added to see how fewer inputs would affect performance of the reservoirs. Because in this setup only part of the reservoir received an input connection, it will be referred to as the “partial input” setup.

It should be noted that “partial input” does not mean that only part of the input pattern was presented. In the “full input” setup, multiple copies of the input pattern were sent to the reservoir for each pattern presentation. In the “partial input” setup, the input pattern was sent to the reservoir roughly once (depending on the outcome of random connections). In other words, usually, the full pattern was still seen by the reservoir with the “partial input” setup.

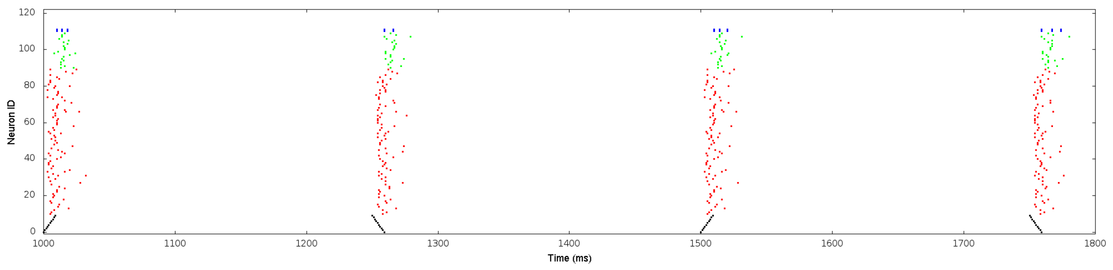
Chapter 4

Results

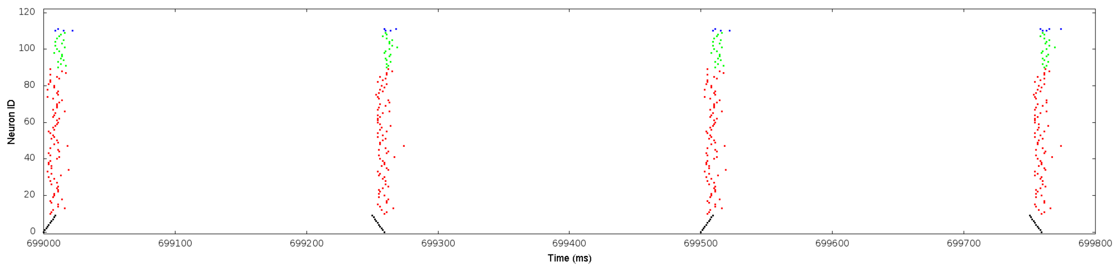
In this chapter, the results of the previously described experiments will be presented. First, the results of our model will be compared to those of the original PM model by Paugam-Moisy et al. [16]. Then, a comparison will be made between the performance of delay adaptation and delay selection. Lastly, the results obtained when using different reservoir topologies (unconnected, random, and small-world) will be compared.

When using delay adaptation and a random reservoir, our model differed from the original PM model mainly by the use of a more biologically plausible neuron and STDP function. Despite these differences, the two models performed very much alike. In [16], success rates (the percentage of correct output firings) of 96.8% for classification of digits 1 and 9, and 80.7% for digits 5 and 8 were reported. Our model reached 96.2% and 80.2% respectively. For abstract patterns with 8 ms noise, our implementation even performed a bit better with a success rate of 88%, compared to 81% for the PM model. Aside from performance, the firing behaviour of the two models was also very similar, as becomes clear when comparing the firings in Figure 4.1 to those presented in [16]. These similarities indicate that the implementation of our model was indeed close to the original, and that the results reported by Paugam-Moisy et al. are reproducible, even with a different neuron model and STDP function.

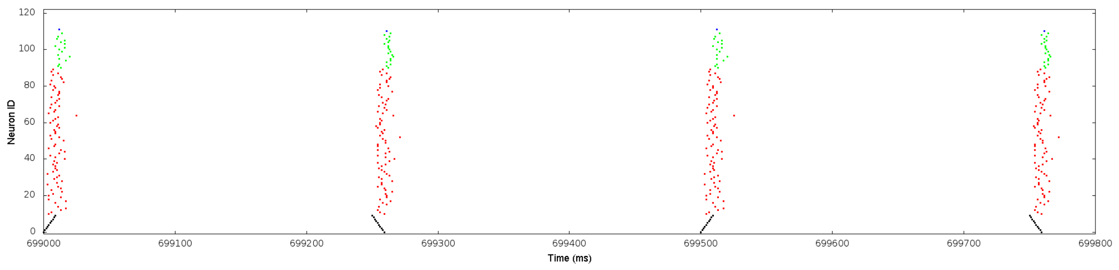
In our first experiment, the performance of delay adaptation was compared to that of delay selection. As can be seen in Table 4.1, delay selection worked, albeit with lower performance in most cases. For the easier tasks of classifying abstract patterns with 0 or 4 ms noise, both learning methods performed roughly the same. However, when input noise reached 8 ms, or when handwritten digits were to be classified, delay adaptation consistently outperformed delay selection by about 10%.



(A) Before learning with delay adaptation



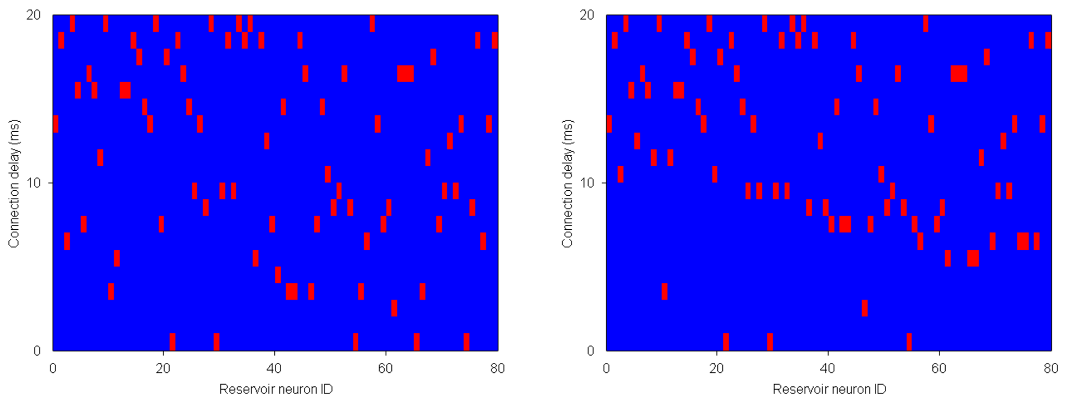
(B) After learning with delay adaptation



(C) After learning with delay selection

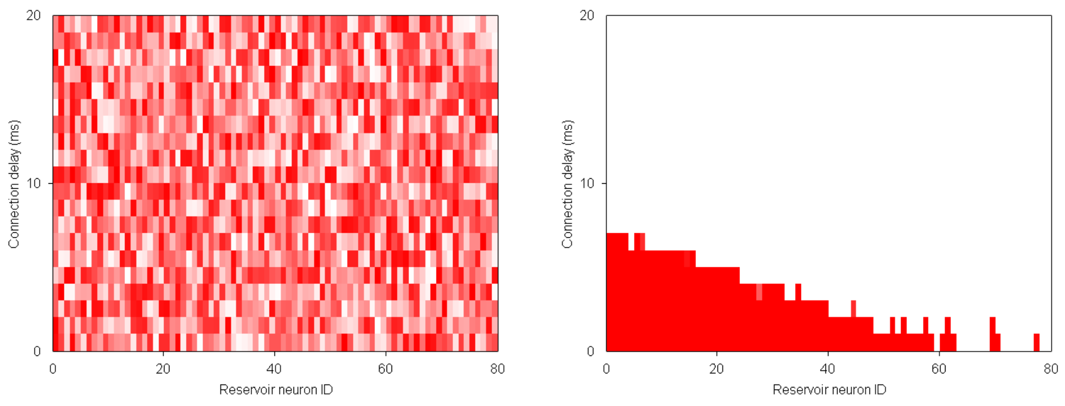
FIGURE 4.1: Firing times of neurons during simulation. A black dot indicates the firing of an input neuron, red an excitatory reservoir neuron, green an inhibitory reservoir neuron, and blue an output neuron. Before learning, the output neurons fire synchronously. After learning with delay adaptation, they fire in a different order for each input class. After learning with delay selection, a different output neuron fires for each class. It can also be seen that, after learning, reservoir activity lasts somewhat shorter than it did before.

The difference in performance between delay adaptation and delay selection can be explained by examining the resulting output connections. These are shown in Figure 4.2 and 4.3, respectively. There it can be seen that the resulting delays for the two methods were quite different. With the setup used for these figures, the ideal delay configuration would be displayed as a diagonal red line. With delay selection however, we see what looks like a red triangle. This is because the connections from each reservoir neuron with delays below optimal, were also consistently potentiated. With delay adaptation, a diagonal line can be discerned, but the delays are far from perfect (many delays are too high, especially). This indicates that the delay adaptation algorithm could be improved upon as well, even though it outperformed delay selection.



(A) Before learning (B) After learning

FIGURE 4.2: Result of delay adaptation. Here, connections from all reservoir neurons to one output neuron are shown, before and after learning. A red square indicates that a reservoir neuron (x-axis) is connected to the output neuron with that delay (y-axis). A blue square indicates that no such connection exists. To be able to interpret the result, abstract patterns without noise were used here, and input neurons were connected to the reservoir orderly instead of randomly, with 8 connections per input neuron. A diagonal line of potentiated connections can be seen after learning, which reflects the abstract pattern to which the output neuron has learned to respond.



(A) Before learning (B) After learning

FIGURE 4.3: Result of delay selection. Here, connections from all reservoir neurons to one output neuron are shown, before and after learning. A red square indicates that a reservoir neuron (x-axis) is connected to the output neuron with that delay (y-axis) and with maximum weight. A white square indicates such a connection exists, but that its weight is zero. To be able to interpret the result, abstract patterns without noise were used here, and input neurons were connected to the reservoir orderly instead of randomly, with 8 connections per input neuron. The potentiated connections after learning reflect the abstract pattern to which the output neuron has learned to respond.

Learning method	Abstract 0 ms	Abstract 4 ms	Abstract 8 ms
Delay adaptation	100 (±0.00)	98.24 (±0.33)	88.20 (±0.74)
Delay selection	100 (±0.00)	98.69 (±0.40)	79.23 (±1.21)

(A) Abstract patterns

Learning method	Digits 1 9	Digits 5 8
Delay adaptation	96.23 (±0.19)	80.20 (±0.88)
Delay selection	87.40 (±1.13)	71.97 (±1.59)

(B) Digit patterns

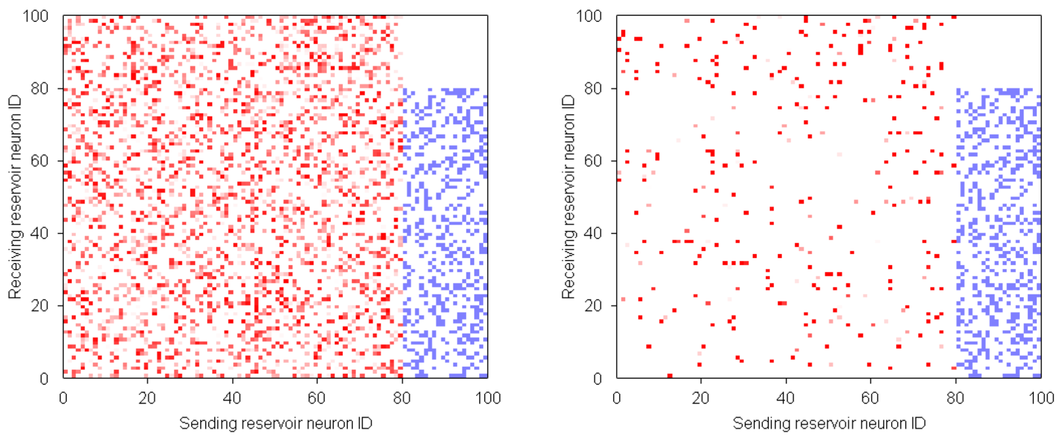
TABLE 4.1: Performance of delay adaptation and delay selection. For each learning method and task, the success rate is reported. This is the percentage of correct output firings, averaged over twenty simulations. The standard error of the mean is noted within parentheses. The top table shows performances for the classification of abstract patterns with 0, 4, and 8 ms noise. The bottom table shows the results for the classification of digits 1 and 9, and 5 and 8. It can be seen that delay selection works, but often performs worse than delay adaptation.

Reservoir type	Digits 5 8	Abstract 8 ms full	Abstract 8 ms partial
Unconnected	81.88 (±0.57)	88.53 (±0.58)	49.34 (±6.74)
Random	79.71 (±0.61)	87.61 (±0.72)	76.84 (±1.64)
Watts-Strogatz	80.46 (±0.87)	89.00 (±0.50)	76.88 (±1.60)

TABLE 4.2: Performance of the unconnected, random, and Watts-Strogatz reservoirs. For each reservoir and task, the success rate is reported. This is the percentage of correct output firings, averaged over twenty simulations. The standard error of the mean is noted within parentheses. Delay adaptation was used here for the delay learning method. In the last two column headers "full", and "partial", refer to the amount of reservoir neurons receiving input. It can be seen that the reservoirs performed roughly the same for digit classification, and abstract pattern classification with full input. With partial input, performance dropped, especially for the unconnected reservoir.

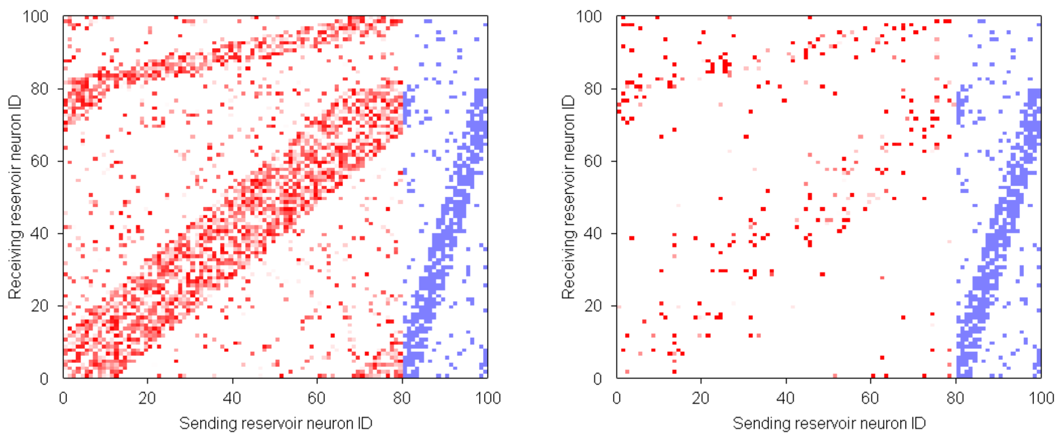
In the second experiment, three different reservoirs were tested. Here, delay adaptation was used for all tasks. The performance of the random reservoir, which was used in the original PM model as well as in the first experiment, functioned as a point of reference. The unconnected reservoir was expected to perform worse, and the small-world Watts-Strogatz reservoir was expected to perform better than the random reservoir, due to the number of polychronous groups that could be formed in each of them. However, as can be seen in Table 4.2, this hypothesis was incorrect. For the classification of digits or abstract patterns in the "full input" setup, the model performed roughly the same with all three reservoirs. In the "partial input" setup, where only part of the reservoir received input connections, performance was lower for all reservoirs. There, the random and WS reservoirs performed the same. The unconnected reservoir performed exceptionally poorly however, due to the fact that output neurons often did not fire.

One possible explanation for the fact that all three reservoirs performed nearly the same, would be that STDP had caused their topologies to become very similar. A look at Figure 4.4 and 4.5 shows that this is true to some degree. These figures show that the number of potentiated connections in the (connected) random and WS reservoirs became lower as a result of STDP, thereby making them more similar to the unconnected reservoir, which contains no connections at all. Still, once weights were stable, enough potentiated connections were left in the connected reservoirs to be able to clearly distinguish the three reservoirs from each other. It is therefore likely that their similar results were caused (in part) by something else. This, and other matters regarding the results, will be discussed in the following chapter.



(A) Before learning (B) After learning

Figure 4.4: Weights of connections in a random reservoir receiving full input. Red indicates an excitatory connection, and blue an inhibitory one. Here neurons 0-80 were excitatory, and neurons 80-100 were inhibitory. STDP depressed many excitatory connections, but not all. Inhibitory connections were static. Weights were stable before the learning phase was over.



(A) Before learning (B) After learning

Figure 4.5: Weights of connections in a Watts-Strogatz reservoir receiving full input. Red indicates an excitatory connection, and blue an inhibitory one. Here neurons 0-80 were excitatory, and neurons 80-100 were inhibitory. STDP depressed many excitatory connections, but not all. Inhibitory connections were static. Weights were stable before the learning phase was over.

Chapter 5

Discussion

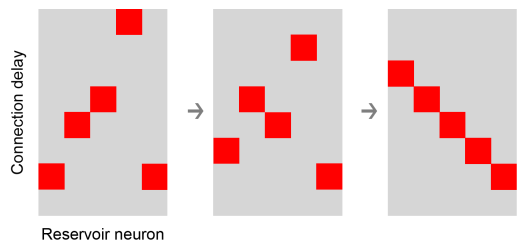
The results presented in the previous chapter firstly showed that the delay adaptation algorithm can be replaced by the more biologically plausible delay selection, but at the cost of some performance. Secondly, it could be seen that in the full-input setup (with nearly all reservoir neurons receiving input connections), all three reservoirs performed the same, regardless of their topology. Lastly, the reservoirs with partial-input did show some differences in performance, but all performed worse than in the full-input scenario. Explanations for these findings will be discussed separately in the following sections.

5.1 Delay learning methods

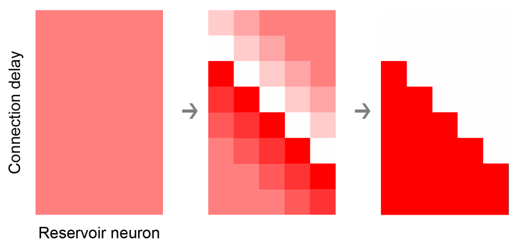
It can be seen that, for the easier tasks where noise was limited, both delay learning methods performed well. As difficulty increased however, the performance of delay selection started to lag behind that of delay adaptation. Two explanations may be given for this difference in performance.

The simplest explanation is that the parameters for both learning methods may not have been equally fine-tuned. Parameters for delay adaptation were converted from the model by Paugam-Moisy et al., who may have done much optimisation, while those for delay selection were chosen by us based on initial testing. Due to slow simulations, a multitude of parameters, and limited time, we were unable to optimise our implementation of delay selection very rigorously.

Secondly, the difference in performance may have been caused by the different delay configurations that were the result of each delay learning method. While delay adaptation resulted in one connection with a (hopefully) optimal delay, delay selection caused multiple connections with different delays to be potentiated. In other words, the result



(A) Delay adaptation



(B) Delay selection

FIGURE 5.1: Schematic display of the learning process for delay adaptation and delay selection. Here, it is assumed that reservoir neurons repeatedly fire in an abstract (sequential) pattern, without noise. Each square represents a connection to an output neuron, from a presynaptic neuron (x-axis), with a delay (y-axis), and a weight (color). The color gray indicates that there is no connection from that reservoir neuron with that delay, white indicates that a connection does exist, but with zero weight, shades of pink indicate intermediate weights, and red stands for a connection with maximum weight. Delay adaptation modifies the delays of connections, while delay selection modifies their weights, through STDP. In the end, delay adaptation keeps one connection per reservoir neuron, while delay selection ends up with multiple potentiated connections, due to the width of the STDP function.

of delay adaptation was a “hard” single value, while that of delay selection was a “fuzzy” distribution of values. The difference is illustrated in Figure 5.1, for a situation without noise.

There are two causes for the potentiation of multiple connections by delay selection. The first is the shape of the learning rule used to modify weights. We used an STDP function, which, due its wide range of positive values, would potentiate multiple connections. Instead of this, a narrow learning rule could be used that potentiates only a single connection. In our preliminary tests however, this resulted in lower performance.

Noise is the second reason why delay selection will potentiate multiple connections. If there is noise, delay adaptation will result in a single delay that may deviate from the optimal value, while delay selection will result in multiple partially potentiated delays that are near it. This can also be observed from the results presented by Natshlager and Ruf (1998) [15]. There, a narrow learning rule was used in combination with noisy input patterns, and as a result, multiple connections were potentiated.

5.2 Reservoirs with full input

In the main setup that was also used in the PM model, where on average all reservoir neurons received input, it was seen that all three reservoirs performed roughly the same. Even the unconnected reservoir without any internal connections performed as well as the connected (random and Watts-Strogatz) reservoirs.

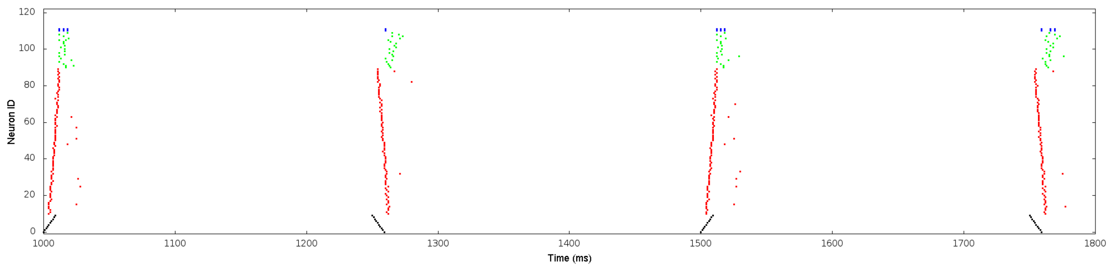
The reason for this was not obvious when we looked at the reservoirs' different topologies. It becomes more clear however, when we examine the firing activity in each reservoir. Firings there can be caused by spikes from input neurons (input activity), or from other reservoir neurons (internal activity). Internal activity is never seen in the unconnected reservoir, because it contains no internal connections. Interestingly though, the amount of internal activity in the connected reservoirs was initially also very limited, and disappeared almost entirely after learning through STDP. This can be seen clearly in Figure 5.2, where input neurons were connected to the reservoir orderly (not randomly), so that the internal activity could be more easily distinguished. Since internal activity was so sparse, all three reservoirs exhibited essentially only (the same) input activity, and therefore performed almost identically.

Internal activity in the connected reservoirs was limited, because almost all reservoir neurons received input connections. Before learning, their refractory period discouraged neurons from firing due to input spikes, and then again due to reservoir spikes. On top of this, after learning, such double firings were almost entirely prevented by the way that weights had been modified by STDP.

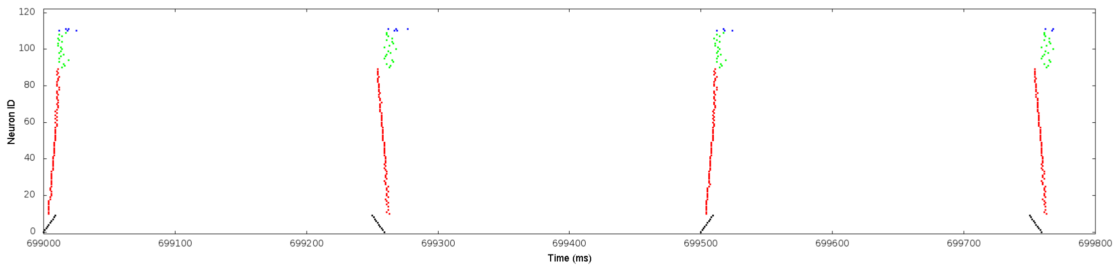
To illustrate how this works, imagine two reservoir neurons A and B, with a connection from A to B with a 3 ms delay. Suppose both neurons fire simultaneously due to spikes from input neurons. Three milliseconds later, a spike from A arrives at B, and the weight of this connection is lowered because B had just fired. If the weight was (still) high enough though, this spike could cause B to fire again, and the weight of the connection would be increased again. However, because in our STDP function depression is larger than potentiation, this weight would still eventually go down, and prevent B from firing due to A. Both neurons then fire only when they receive spikes from input neurons.

Because internal activity was almost non-existent (due to the combination of full input, refractory periods, and STDP), the reservoir exhibited no short-term memory caused by recurrent connections, and performed no (non-linear) transformations to make classification easier. Input patterns were simply sent unaltered through the reservoir to the output neurons. This means that the reservoir was of little use, and performance would be the same if input neurons had been connected directly to the output neurons.

This finding conflicts with the report by Paugam-Moisy et al. (2008) [16], where they highlighted the importance of using a reservoir with STDP, in combination with the delay adaptation learning method. There it was stated that the reservoir contained polychronous groups that were active only for particular classes of inputs, and that the detection of these active polychronous groups was what made delay adaptation work.



(A) Before learning



(B) After learning

FIGURE 5.2: Firing times when using ordered connections between input and reservoir neurons. Here, each input neuron was connected to 8 consecutive reservoir neurons. This differs from the normal setup where these connections were random. In this figure, a black dot indicates the firing of an input neuron, red an excitatory reservoir neuron, green an inhibitory reservoir neuron, and blue an output neuron. It can be seen that almost all reservoir activity is caused directly by the input neurons. The few other firings in the reservoir disappear entirely after learning with STDP.

Our results indicate, however, that the active polychronous groups in the experiments by Paugam-Moisy et al. were not the cause of their network's performance, but merely a side-effect of its workings. As mentioned before, after STDP had run its course, (nearly) all activity in the reservoir was caused directly by input firings. This means that any active polychronous groups must have actually been small parts of the input pattern that happened to match connection delays in the reservoir. For the abstract input patterns, these polychronous groups (or these parts of the input) were active only for one class of input, because the input patterns had no parts in common and did not contain noise. For the digit patterns which were noisy and could have parts in common, the charts in [16] show that active polychronous groups were actually not very good indicators of input class at all, with no PG's that were active for more than 25% of the patterns for a class, and many PG's that were active for both classes.

5.3 Reservoirs with partial input

In the setup where only part of the reservoir received input, all reservoirs performed worse than in the full-input setup. Here we will first explain the low performance of the unconnected reservoir, and then that of the connected (random and WS) reservoirs. Finally, we will discuss possible reasons for the similar performance of the random and WS reservoirs in this setup.

As mentioned before, the unconnected reservoir performed very poorly. The reason for this is that, due to the lack of internal connections, the majority of reservoir neurons never fired. Only the few neurons receiving input connections could fire. As a result, there was often not enough reservoir activity to make the output neurons fire when needed. We expect that performance could have been improved significantly by increasing the weights of connections to the output neurons.

The connected (random and WS) reservoirs performed better than the unconnected reservoir, but still worse than they did in the full-input scenario. Here, with partial input, internal activity was finally possible. This means that, alongside a copy of the input pattern presented by the neurons that received input connections, the reservoir could also exhibit a possibly complex and non-linear transformation of this input through its other neurons. This means that the total amount of information in these reservoirs should be at least the same as in the full-input scenario, but possibly also in a form that would be easier to classify. Still, performance here was lower. It seems that this must be due to a weakness of the delay learning method.

One limitation of the delay adaptation method was that weights to the output neurons were static. This forced these neurons to respond to a predetermined number of simultaneously arriving spikes. Therefore, patterns in the reservoir consisting of fewer or more spikes, could not be detected as reliably. Delay selection suffered from the same problem. Although weights there were modified through STDP, they were eventually bound to a fixed maximum value, resulting in the same limitation that was caused by static weights. A possible solution to this problem (synaptic scaling) will be discussed in Section 6.1.

Finally, it was seen that with partial input, the random and WS reservoirs still performed the same. It is possible that these reservoirs are simply equally useful for the task for which they were used. However, it could also be the case that the delay learning mechanism was unable to take advantage of the possibly superior internal activity in one of the reservoirs. Further research would be needed to determine if a small-world reservoir could indeed be advantageous in this model.

The following chapter will present our conclusions, as well as suggestions for future work.

Chapter 6

Conclusion

We successfully replicated the reservoir model by Paugam Moisy et al. [16], and tested it on the classification of images of handwritten digits. While using a more biologically plausible neuron model and STDP function, our results were similar to those reported in the original paper.

The original delay adaptation mechanism was successfully replaced by delay selection with STDP, although at the cost of some performance. The lower success rates with delay selection may be caused by sub-optimal parameters, or by the fact that delay selection can result in a less precise delay configuration. Because our implementation of delay selection requires only the existence of multiple pathways with different delays, and the biologically observed STDP mechanism, it can be seen as a biologically plausible alternative to delay adaptation that may be effective enough, depending on the tasks and circumstances.

Surprisingly, it was discovered that the reservoir in our model served no purpose. Using a reservoir without internal connections, or connecting input neurons directly to output neurons, had no negative impact on performance. We believe this must also be true for the model by Paugam-Moisy et al. [16] because, to our knowledge, its functioning is identical. The main reason why the reservoir had no added value here, was that almost all reservoir neurons received input. This, combined with refractory periods and STDP, prevented the reservoir from functioning as it was intended to. Because of this, the use of a small-world Watts-Strogatz reservoir did not affect performance.

When only part of the reservoir received input, the reservoir should have been able to function properly. Unfortunately though, the model performed much worse in this case. This was likely caused by limitations of the delay learning mechanism. Further research with an improved output learning mechanism would be needed, to determine whether a small-world reservoir can be more useful than a random reservoir in this setup.

Although the reservoir in this model failed to meet expectations, we emphasise that this was caused by certain aspects of the model, and that it does not indicate an inherent flaw in the concept of reservoir computing. Many models have performed excellently thanks to the use a reservoir, and the concept therefore still seems very promising.

6.1 Future work

Our findings give rise to several ideas for future research. These will be discussed in the following paragraphs.

Firstly, in section 5.3 it was noted that both the delay adaptation and delay selection methods in our model, had an important limitation. Each method caused the output neurons to become optimised for the detection of patterns with a fixed, predetermined number of spikes. Patterns with fewer or more spikes could not be detected reliably, because they provided too little or too much excitation, resulting in false negative or false positive firings.

It is possible that synaptic scaling could solve this problem. This biologically observed mechanism normalises the weights of a neuron's (receiving) synapses, in order to keep its average firing rate stable [19]. This means that it keeps the sum of the synaptic weights of a neuron constant, while retaining their ratios, so that a neuron can have for example few highly potentiated, or many slightly potentiated connections. In combination with STDP, this should allow a neuron to learn to detect patterns with various numbers of spikes, without firing too often or too little.

We therefore suggest, for future work, that a form of synaptic plasticity could be incorporated into the reservoir model. With normalised weights to output neurons, pattern detection should be independent of the amount of reservoir activity, and therefore more effective.

Secondly, we believe that for delay selection, the function of multiple delay lines between neurons can actually be provided by the reservoir itself, if it is large enough. This means that with delay selection (as already with delay adaptation), only a single connection would be needed between each reservoir and output neuron. The delay lines would then consist of indirect connections that pass through one or more reservoir neurons before reaching an output neuron. Future experiments could indicate if this approach is feasible, or whether such indirect delay lines would work differently than direct connections. The resulting model, however, should no longer be called a reservoir model, since the learning mechanisms for reservoir and output neurons would be identical.

Finally, some other interesting improvements could be made to the model. For one, the input could be encoded through neurons with receptive fields, as was done in [2]. Also, the unsupervised delay selection could be turned into a supervised learning mechanism, by modulating STDP with dopamine when the desired output is given and a reward is presented [10].

These suggestions for modifications of the model, are all based on biological findings. The results will therefore hopefully bring us closer to an understanding of the complex system that is called the brain.

Appendix A

Appendix

A.1 Software implementation

The first version of our software, was written in Java. The initial goal was to replicate the results of the PM model by Paugam-Moisy et al. Therefore, we used the same network structure, delay learning mechanism, STDP functions, and neuron model. With this model, we obtained nearly identical results to those reported in [16].

The next step was to modify this program to our needs. We would add a different neuron model, STDP function, delay learning mechanism, and different reservoirs. Unfortunately, however, the SRM_0 neuron model in our application could not be replaced with the more biologically plausible Izhikevich model. The problem was that the program was event-based, instead of time-based. This means that membrane potentials were not calculated each time step, but only when needed (when spikes arrived). The event-based mechanism was fast, but not very compatible with Izhikevich neurons, because they can fire also during time steps when no external events (spike arrivals) occur.

We therefore decided to rewrite the program, because modifying the original to use a time-based mechanism would be even more time-consuming. This new application was time-based from the start, and contained all the modifications that we wanted to make to the original model. All results reported in this thesis were obtained with this second program. The new software was written in Golang instead of Java. Golang is a relatively new, compiled programming language that was developed by Google. It was chosen for its speed and useful language features such as first-class functions.

The new program could output data to plain text files, which were then converted into graphs by gnuplot scripts. To inspect networks during and after simulation, several values could be logged and graphed. These include membrane potentials, firing times, weights and delays of connections, weight distributions, success rates of output neurons, and more.

We found out, that when using the Izhikevich neuron model, one should calculate membrane potentials for each 1 ms time step, in multiple smaller increments (such as four steps of 0.25 ms). This is needed for numerical stability, and was also done by Izhikevich in [8]. Although there, two steps were used, we used four. This prevented all erroneous values for membrane potentials in our model, which could occur for example when giving neurons very large input values.

Simulations were run on an Intel Core 2 Quad Q6600 CPU. With this, computation times for experiment simulations varied from 10 to 30 minutes.

A.2 Attempts at supervised delay selection

Two attempts were made to convert the delay selection mechanism to a form of supervised learning. This was desired, because delay adaptation was able to classify input, while delay selection could only cluster it.

The first plan was to modulate STDP with a reward signal when the correct output was given. This was originally discussed and implemented in [10]. However, it soon became clear that, due to the added complexity, we could not successfully complete this within the scheduled time period.

We thought of an alternative method however, and also tried to implement this. The idea was that we could teach an output neuron to fire at a desired time, by initially making it fire at that time with a teaching signal. In other words, during learning, we would provided the network with “example” output firings at the desired times. STDP then potentiated connections with delays that matched the example firing time, and depressed others. In this way, the neuron could eventually learn to fire at the desired time on its own.

The reason why this did not work, was that STDP teaches neurons to fire as early as possible. The output neurons therefore soon forgot to fire at the desired time, and instead learned to fire only at the very start of each pattern presentation. Because of this, we eventually settled with keeping our implementation of delay selection unsupervised.

Bibliography

[1] P. Auer, H. Burgsteiner, and W. Maass. A learning rule for very simple universal approximators consisting of a single layer of perceptrons. *Neural Networks*, 21(5):786–795, 2008.

[2] S. M. Bohte, H. La Poutré, and J. N. Kok. Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks. *Neural Networks, IEEE Transactions on*, 13(2):426–435, 2002.

[3] N. Caporale and Y. Dan. Spike timing-dependent plasticity: A Hebbian learning rule. *Annual Review of Neuroscience*, 31(1):25–46, 2008.

[4] C. W. Eurich, K. Pawelzik, U. Ernst, A. Thiel, J. D. Cowan, and J. G. Milton. Delay adaptation in the nervous system. *Neurocomputing*, 32:741–748, 2000.

[5] D. Feldman. The spike-timing dependence of plasticity. *Neuron*, 75(4):556–571, 2012.

[6] Y. He, Z. J. Chen, and A. C. Evans. Small-world anatomical networks in the human brain revealed by cortical thickness from MRI. *Cerebral Cortex*, 17(10):2407–2419, 2007.

[7] J. J. Hopfield. Pattern recognition computation using action potential timing for stimulus representation. *Nature*, 376(6535):33–36, 1995.

[8] E. M. Izhikevich. Simple model of spiking neurons. *Neural Networks, IEEE Transactions on*, 14(6):1569–1572, 2003.

[9] E. M. Izhikevich. Polychronization: Computation with spikes. *Neural Computation*, 18(2):245–282, 2006.

[10] E. M. Izhikevich. Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral Cortex*, 17(10):2443–2452, 2007.

[11] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148, 2001.

[12] M. Lukoševičius, H. Jaeger, and B. Schrauwen. Reservoir computing trends. *KI-Künstliche Intelligenz*, 26(4):365–371, 2012.

[13] W. Maass. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

[14] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[15] T. Natschläger and B. Ruf. Spatial and temporal pattern analysis via spiking neurons. *Network: Computation in Neural Systems*, 9(3):319–332, 1998.

[16] H. Paugam-Moisy, R. Martinez, and S. Bengio. Delay learning and polychronization for reservoir computing. *Neurocomputing*, 71(7):1143–1158, 2008.

[17] B. Schrauwen, D. Verstraeten, and J. V. Campenhout. An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the 15th European Symposium on Artificial Neural Networks*, pages 471–482, 2007.

[18] X. Tao and H. E. Michel. Data clustering via spiking neural networks through spike timing-dependent plasticity. In *IC-AI*, pages 168–173, 2004.

[19] G. G. Turrigiano. The self-tuning neuron: synaptic scaling of excitatory synapses. *Cell*, 135(3):422–435, 2008.

[20] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout. Isolated word recognition with the liquid state machine: a case study. *Information Processing Letters*, 95(6):521–528, 2005.

[21] P. E. Vertes and T. Duke. Effect of network topology on neuronal encoding based on spatiotemporal patterns of spikes. *HFSP Journal*, 4(3-4):153–163, 2010.

[22] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.