Activity dependent delay learning in spiking neural networks

Jørgen J. Farner

Report submitted as part of the
Master in Applied Computer and Information Technology (ACIT)
60 credits

Department of Computer Science
Faculty of Technology, Art and Design

OSLO METROPOLITAN UNIVERSITY

Spring 2022

Activity dependent delay learning in spiking neural networks

Jørgen J. Farner

Activity dependent delay learning in spiking neural networks

http://www.oslomet.no/

Printed: Oslo Metropolitan University

Abstract

Several observations indicate activity dependent changes in the propagation velocity of action potentials in biological neural networks. These changes are believed to be deliberate mechanisms in the brain used to mediate the delays of the connections between neurons for functional purposes. These observations have created a great interest from the research community in the role of neuronal delays. In order to explore these mechanisms, a novel delay learning mechanism has been developed that adjusts delays based on local activity. The main aspect of the learning method is to align the delays of pre-synaptic spikes that are causally related to a post-synaptic spike. The theory is that this will consolidate activity that is related to similar inputs, consequently allowing the network to separate inputs into classes. To accommodate for this method, a coding framework that allows for local delay learning in spiking neural networks was created. A novel method for detecting polychronous groups for the purpose of performing classification tasks was also developed. The delay learning method was applied to various network topologies, and the learning method showed a significant performance improvement when applied to feed-forward networks over identical static networks. The performance was measured on a classification task involving inputs encoded based on relative spike latencies. It was also shown that using rate-coded inputs leads to diverging and unstable delays, indicating that this might not be an appropriate encoding method for this temporal learning mechanism. Finally, the learning mechanism was applied to feed-forward topologies to perform classification on images from the MNIST dataset. The plastic networks showed a significant performance increase over identical static networks.

Acknowledgments

I would like to thank Stefano Nichele for his support throughout the thesis process and for sharing his vast knowledge of artificial intelligence which has been of great help. I would also like to thank Ola Huse Ramstad for his invaluable insight into the biological aspects of neural networks. Finally I would like to extend a special thanks to Kristine Heiney, who has patiently supported, encouraged and guided me every week throughout this one and a half year process. This would not have been possible without your help. It has been a tremendously rewarding experience to work with these three great researchers.

Contents

Chapter 1

Introduction

Curiosity about the inner workings of the brain is an age-less endeavour that has perhaps been present since the dawn of complex primate consciousness. For the vast majority of this time the brain has been viewed as a black box, hiding the secrets about its intricate design. With the emergence of advanced technology and research methods that accompanied the twentieth century, we are beginning to unveil the answers to our many questions regarding the human brain. At the same time computer scientists discovered the possibility of recreating the mechanisms and structures found in the brain as a computational tool, and thus the research field of artificial intelligence emerged.

From its humble beginnings, artificial intelligence has come a long way. The research field started off with the Threshold Logic Unit (TLU) [30], capable of approximating simple boolean logic, and has now evolved to the modern deep neural networks of today, capable of generating photo realistic images from text input [52]. Artificial neural networks (ANN) have been implemented to solve numerous real-world tasks that are either laborious for humans to perform or in many cases impossible. Still, the goal of achieving general intelligence in an artificial network is distant. However, this shouldn't deter us from exploring new methods and ideas to make incremental improvements to our current models.

A significant limitation of conventional ANNs is their narrow domain and lack of generalizability. Although the brain consists of several specialized sub-networks that are structured to perform specific tasks, the brain also consists of large areas that can perform more general tasks. This flexibility is a desired trait to be reproduced in ANNs, and with this motivation in mind a new paradigm is emerging with an increased focus on drawing inspiration from biological networks. With this change of view, spiking neural networks (SNN) have made their way into mainstream research and stands as a possible successor to sigmoidal networks. Another significant reason for a lot of current research pertaining to SNNs is their ability to be implemented at a hardware level giving them a notable increase in efficiency over their ANN counterparts.

The real-values output of the conventional ANNs can be considered an abstraction of spike-rates [15]. However, contrary to the long-held notion

that neurons solely communicate with the firing-rate of action potentials, many researchers current position is that the firing-rate is an insufficient encoding method to explain all communication observed in biological neural networks. Instead, it is believed that the precise timing of action potentials plays a crucial role in the brains efficient processing ability, and consequently the propagation delay between neurons is a vital part of this. It is therefore logical to assume that the tuning of these delays could be a crucial aspect of how the brain learns.

SNNs differ from conventional ANNs in primarily three aspects; activation functions are replaced with spiking neurons, real-valued outputs are replaced with sequences of spikes and lastly the introduction of a temporal aspect by replacing the layer-wise matrix operations of conventional ANNs with discrete time-steps where neurons can produce action potentials asynchronously, and these action potentials travel through connections. It is believed that this temporal aspect opens up the possibility for SNNs to exceed the performance of conventional ANNs both in terms of efficiency and in terms of computational capabilities. However, current implementations of SNNs rarely outperform conventional ANNs, and a lot of work still remains to uncover their true potential. As part of this process, the role of propagation delays and how incorporating delay learning may affect the networks computational capabilities, must be explored.

1.1 Motivation

There are several mechanisms and properties that affect the velocity at which an action potential travels, like axon diameter, temperature and ion channel density. Another important factor is the presence of a myelin sheath that insulates the axon and increases the velocity of the action potential [37]. The observed delay can therefore be considered as the result of different mechanisms and properties that affect the propagation velocity along with the physical length of the axon and dendrite connecting the two neurons. However, the propagation delays do not necessarily appear to be static at all times according to several observations [4], [13], [14], [28], [7], which describe activity dependent changes to the propagation velocity of action potentials. These observations indicate that there exists deliberate mechanisms in biological neural networks that mediate the delays between neurons with a functional purpose. From a computational point of view Izhikevich [21] demonstrated that optimizing the delays of a SNN increases the amount of information that can be represented. These observations constitute the motivation for this project and as a step towards understanding the role of activity dependent delay plasticity and how it can increase the computational capabilities of neural networks, this project developed and analyzed a novel local delay learning mechanism.

1.2 Problem Statement

Current SNNs are mainly considered for research purposes rather than solutions to real-world problems as they rarely if ever outperform conventional ANNs. Especially when implemented on conventional hardware, the computational efficiency of SNNs is quite inferior to that of conventional ANNs. This is in stark contrast to the extreme efficiency with which the biological brain operates, and it is therefore believed that we have not seen the full potential of SNNs.

Although various learning methods, both supervised and unsupervised, that strengthens and weakens connections have been applied to SNNs, there exists very little work directly associated with the training of delays, and practically no work has been done on developing local learning methods for delay training. It is this gap of knowledge that this project attempts to address by improving network performance on classification tasks through the implementation of a novel delay learning method that mediates delays based on local knowledge.

Chapter 2

Background

This section will present information that is relevant for this work. A thorough exploration of the current state of knowledge within the relevant fields provides the background information that is needed in order to create new machine learning models with some improvement or novelty. These fields include ANNs, neuron models, learning methods and coding schemes.

During the project a novel local learning mechanism applied to SNNs was developed, therefore an introduction to the principal features of SNNs and a comparison to general neural networks will be given.

Any neural network consists of smaller computational units or artificial neurons. In the context of SNNs, these computational units are often referred to as neuron models. The most common neuron models will be discussed and compared, highlighting their strengths and weaknesses.

As the main focus of the project is the development of a learning mechanism, it is important to provide an understanding of the biological learning mechanisms present in the brain, along with existing learning frameworks applied to ANNs.

In biological neural networks, information is believed to be encoded by various coding schemes, depending on the area of the brain in question. These coding schemes differ in their efficiency and robustness. Likewise, this project applies a specific coding scheme to convert information into spikes, with which the network can process and perform computations on. It is therefore important to explore some of the existing coding schemes that are frequently implemented in SNNs.

In order to run simulations of SNNs, a coding framework that explicitly allows for implementing self-defined delay learning methods must be used. A section of this background chapter is therefore dedicated to the process of evaluating existing frameworks.

The final section of the background chapter will explore related works which either implement learning mechanisms on delays in SNNs or otherwise explicitly depend on delays to perform computation.

2.1 Artificial Neural Networks

The term artificial neural network refers to any computational framework that relies on interconnected nodes or artificial neurons loosely inspired by real biological neurons, and encompasses a large variety of different implementations of this definition. However, if one groups them by their temporality, one can separate the networks into two major categories, namely the spiking neural networks which relies on the spiking neuron and the conventional neural networks which relies on what is often called the sigmoidal neuron. The first category includes a temporal aspect where information travels along connections between neurons as discrete spikes. The latter category, however, transfers information immediately in a step-wise manner from one neuron to the next, and the information takes the form of a real number. In this section, these two types of artificial neural networks will be described and compared.

2.1.1 Conventional Artificial Neural Networks

The group of computational frameworks the fall under what can be referred to as conventional artificial neural networks, rely on relatively simple mathematical units which operate on real values and lack the time component of SNNs. With its growing number of variations, these frameworks have virtually become synonymous with machine learning in recent years. However, the idea of taking inspiration from biological brains and creating a network of artificial neurons is not a novelty and has been around for some time. The limiting factor that prevented NNs from becoming widely used computational frameworks was the need for large datasets required for training the models. This requirement posed a great challenge in the early days of conventional NNs since large data sets were not readily available. This has since changed with the global adaptation of the internet, and the steady transfer from analog to digital storage of information

The conventional ANNs of today are able to exceed human capabilities in various tasks, but their usefulness was quite limited during their humble beginnings. In 1943 McCulloch and Pitts [30] proposed the first artificial neuron. Their proposed neuron model receives weighted inputs from all of its pre-synaptic neighbours, and if the summed input is greater than a fixed threshold, the neuron outputs a binary value of 1. If the threshold is not exceeded, the output is 0. The weights of the connections determine if it is providing an excitatory or inhibitory input. The input is excitatory if its value is above zero, while it is inhibitory if its value is below zero. There is no temporal aspect to the model, as the output of one neuron is immediately received at the post-synaptic neuron during the next computational step. The McCulloch and Pitts neuron can be referred to as a threshold function and can be seen in figure 2.1.

$$f(x) = \begin{cases} 1 & \text{if } \sum_{i=0}^{m} w_{ij} \cdot x_i > \psi \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

$f(x)$ is the output of neuron j, ψ is the threshold of the neuron, w_{ij} is the weight of the connection from neuron i to neuron j and x_i is the output of neuron i. This threshold logic allowed the McCulloch and Pitts neuron to perform boolean operations when configured together in networks. However, they lacked the ability to separate non-linearly separable input, limiting their usability. In simple term, if one considers 2-dimensional data containing two distinct classes, the data can be considered linearly separable if it is possible to draw a straight line, or first degree polynomial, that separates all instances of the two classes. If the data is not linearly separable, a more complex function is needed to separate it.

Perhaps the biggest shortcoming of the McCulloch and Pitts neuron was its inability to adjust weights or thresholds, in other words it lacked the ability to learn. The models needed to be manually configured for a specific task like performing a boolean AND operation.

The invention of the Perceptron by Rosenblatt in 1958 [38] introduced several improvements to the McCulloch and Pitts neuron. A crucial improvement was the inclusion of supervised learning which allowed a network of Perceptrons to be trained for specific tasks, although the task was limited to binary classification. The Perceptron also suffered from the inability to separate non-linearly separable input if configured in a single layer.

These issues with the early neuron models were addressed with the modern artificial neuron which exchanged the threshold function with a non-linearity called an activation function such as the Sigmoid function seen in eq. 2.2.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

The output of these neurons is a continuous set of values, $f(x) \in \mathbb{R}$, and as a complete network their ability to approximate any continuous function has been demonstrated [17]. Perhaps the most common versions of the modern ANN, is the recurrent neural network and the convolutional neural network, which both implement these computational units and have become the norm for solving classification tasks related to sequential data and image recognition respectively.

A limitation of trained ANNs comes from the fact that they are approximating a specific function. An ANN can essentially be thought of as a universal function approximator, and through a substantial training process its weights are adjusted until it has become an approximation of a very specific function that does something very specific. Without retraining, the model cannot properly perform other tasks.

Another limitation of ANN's is that they propagate information from input layer to output layer in a step-wise manner, thus the intra-neuronal delay caused by the variable propagation speed of the action potentials (AP) is neglected. Research into the information processing in certain areas of the brain [35] has proven spike rate to be insufficient as the sole encoder of information present in biological computational processes. Perrett, Rolls and Caan [35] demonstrated that humans can perform simple visual analysis in less than 100ms, a process that requires the information

to propagate through at least 10 synaptic layers from the retina to the temporal lobe, with the involved neurons firing at a rate typically below 100Hz. Considering that there must be a 20-30ms time window to allow for fire rate sampling, their work indicates that there is a clear discrepancy between the observed time window for the visual processing and the time-scale at which rate-coding would accomplished said task. It has also been observed that neurons are able to produce spikes with very precise timing [5], [11]. This observation led scientists to believe that the information processing in the neurons is far more complex and that the precise timing of spike-arrival at the post-synaptic neuron might be a significant part of the information encoding. It is likely that this lack of temporal encoding in conventional neural networks limits their computational abilities and prevents them from achieving any form of general intelligence.

Creating networks that leverage some temporal coding might lead to processing capabilities that exceeds that of ANNs in various ways. It has also been proven that when SNNs are implemented in hardware, referred to as Neuromorphic computing, the energy efficiency is drastically increased [27]. This realization has led to a recent shift in focus among researchers towards exploring spiking neural networks. The next section will give an overview of the most important aspects of SNNs and how it differs from conventional neural networks.

2.1.2 Spiking Neural Networks

SNNs takes the biological inspiration found in conventional ANNs a step further by including a temporal component in addition to the inclusion of more biologically realistic neuron models. When considering their computational units, SNNs have been referred to as the third generation of Neural Networks [29], with the first being the Perceptron or McCulloch-Pitts neuron and and the second generation being the continuous activating neurons of modern NN's as seen in figure 2.1. It is worth to note that the Perceptron and the McCulloch-Pitts neuron are quite similar and although they differ in certain aspects, they can both be considered Linear Threshold Units (LTU), and for simplicity's sake, LTU will be used for future comparisons to SNNs.

Figure 2.1 neatly portrays the three main aspects in which SNNs differ from the other two generations of ANNs and the biological brain. The first aspect is the individual computational units, or the neurons. As previously mentioned, the artificial neuron of conventional ANNs, sums the input which is then given to an activation function which calculates the output of the neuron. The LTUs function in a similar way, except it uses a threshold function instead of an activation function. Although there are multiple spiking neuron models, their common mechanism is to integrate inputs that arrive in the form of weighted spikes, and if a threshold is exceeded, the neuron produces its own spike. At first glance, there are conceptual similarities between spiking neurons and LTUs, with thresholds that when exceeded leads to a spike or the binary value of one, in the case of the LTU. However, SNNs function in a substantially different way due to their

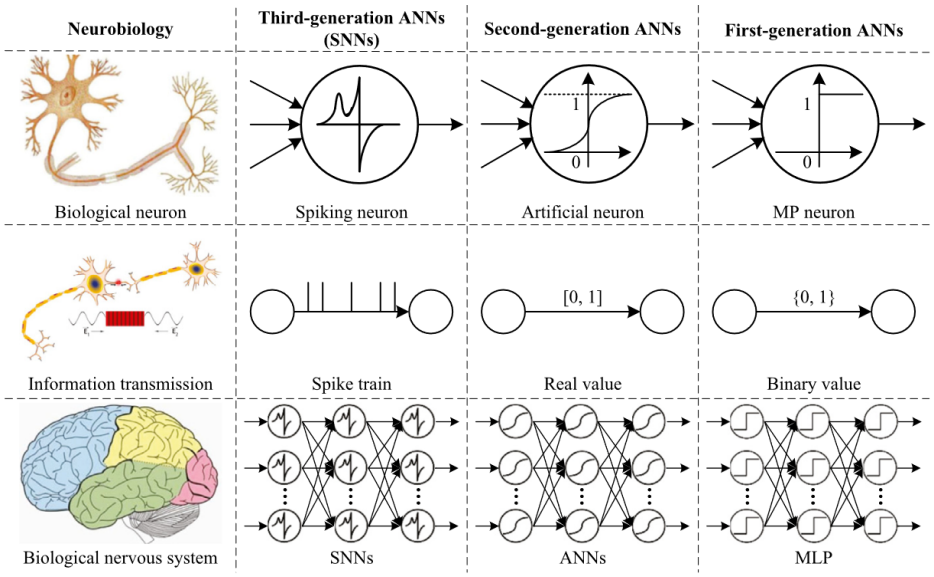

Figure 2.1: A comparison between the three generations of ANNs and neurobiology. The computational units, information transmission and topology is compared [48].

temporal aspect, which leads to the second aspect which is the nature of the neuronal communication. Whereas the LTU and artificial neuron outputs binary or real values, which is immediately received by the post-synaptic neuron, the spiking neuron outputs temporal spikes which travel along the connections and arrive at the post-synaptic neuron at a time dictated by the delay of the connection. The spiking neuron can produce a sequence of spikes. This sequence of spikes is referred to as spike-trains and allows for a substantial amount of information to be encoded either in the rate of spikes or according to some temporal aspect of the spike train, which will be discussed further in section 2.4

The final aspect that differentiates the SNN from the other generations of ANNs is its configuration as a network. In contrast to the previous two generations of ANNs, which are often configured in layered structures or otherwise homogeneously distributed recurrent connections, the SNN can be configured in practically endless ways. Although, the SNN is often used in a feed-forward topology, it is also common to configure it in a more random way. These random configurations of SNNs are often referred to as reservoirs or liquid state machines, which is a sub-category of reservoir computing. Although these configurations are sensitive to the choice of hyper-parameters, such as connectivity, a properly configured reservoir is a very capable computational framework. Such configurations are particularly suitable for sequential data, as the temporal nature of SNNs allows information to persist for some time in the network.

One distant goal of Artificial Intelligence is to create general intelligence that mimics the cognitive abilities of biological brains to some extent. Although conventional ANNs can be applied to a wide variety of problems,

they are very domain specific once they are configured and trained. A large enough collection of these models, might constitute a more complex framework as a whole, but hand-picking models to perform certain tasks results in something that is not very flexible or adaptive and is computationally expensive on current hardware.

The implementation and more specifically the training process of modern Deep Neural Networks (DNN) through methods like error back-propagation (EBP), explained further in section 2.3.3, is computationally expensive and consequently an energy inefficient process. SNN's might be the solution to this problem as they offer the possibility of being implemented at a hardware level, referred to as Neuromorphic computing, where neuronal structures are directly embedded in processor chips in order to emulate real neural circuits. When each neuron and connection of the neural network is implemented as electrical components, the efficiency is drastically increased as each component expends energy only when active. Running neural networks on conventional hardware, on the other hand, requires a lot of energy due to the synchronous global clock which forces all computational units, even ones that are not necessary for the task, to respond [27].

2.2 Neuron Models

Spiking neural networks are inspired by biology and this includes the computational units that they comprise of. Many neuron models have been put forward and there is often a trade-off between creating a biologically plausible model and creating a computationally efficient model. The reason for this is that the biologically plausible models attempt to capture the complexity of biological neurons and implement them in artificial circuits with the existence of several abstraction layers between the code and the hardware. This bottom-up approach leans heavily on the current biological knowledge of neurons and its connections at the expense of usability in larger systems. Another approach, a top-down approach, has been to create a computationally efficient model that produces behavior similar to that of a real cortical network. Figure 2.2 maps some of the most popular neuron models with respect to their biological plausibility and computational efficiency. The biological plausibility is determined by the number of features that the respective models include 2.7. These features are various spiking behaviors and intrinsic properties. The implementation cost is measured in floating point operations per 1ms of simulation.

Since this work focuses on implementing networks with sufficient size that allows for solving complex tasks, the choice of model relied heavily on computational efficiency. However, biological realism is also preferred if the computational penalty is not too severe, as the learning method is also inspired by biological mechanisms. The next subsections will detail the different model types that were deemed relevant either because of their biological aspects or because of their computational efficiency.

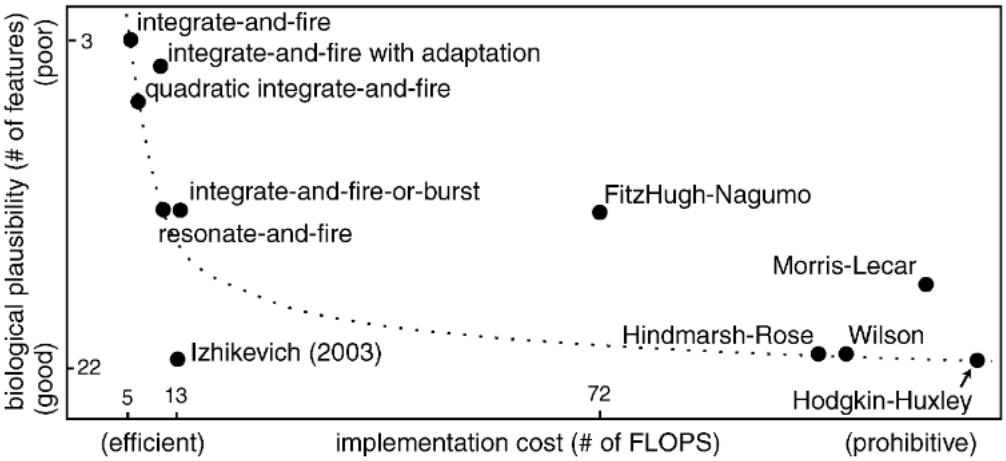

Figure 2.2: An overview of neuron models, their biological plausibility and implementation cost [19].

2.2.1 Hodgkin-Huxley

Although it is one of the most computationally expensive models, the Hodgkin-Huxley model [16] is the most comprehensive and biologically accurate. It details various biological aspects of the neuron in great detail, like the Na^+ and K^+ currents, by representing the neuron as an electrical circuit of components. In order to understand the model, it is necessary to have an understanding of the biological mechanisms that are present in the neuron, axon and dendrites.

The biological neuron is the computational unit of all animal and human brains. These neurons are clustered together to form larger structures that have some functional purpose. A simplified explanation of the intra-neuronal communication is that it is done by emitting action potentials which are electrical signals that propagate along the connections that binds the neurons together. These connections primarily consists of three separate parts, the axon, the dendrites and the synapse. The different parts of the neuron can be seen in figure 2.3. The axon is the output connection of the neuron denoted d in figure 2.3, while the dendrites are the input connections of the neuron, denoted a in figure 2.3. The synapse is the connection between the two. A neuron has typically, but not always, a single axon that conducts its output action potential, while it has multiple dendrites that attaches to other neurons and conducts incoming signals. In addition to the parts already detailed, the neuron consist of a cell body that houses the nucleus which is a central part of the cell that controls the activities of that particular cell. The Myelin-sheath insulates the axon so that ions do not escape through its membrane. The Schwann cells are responsible for wrapping around the axon to create the Myelin-sheath. The Nodes of Ranvier are gaps in the Myelin-sheath that allow for the exchange of ions with the extracellular space which enables the action potential to propagate through the axon.

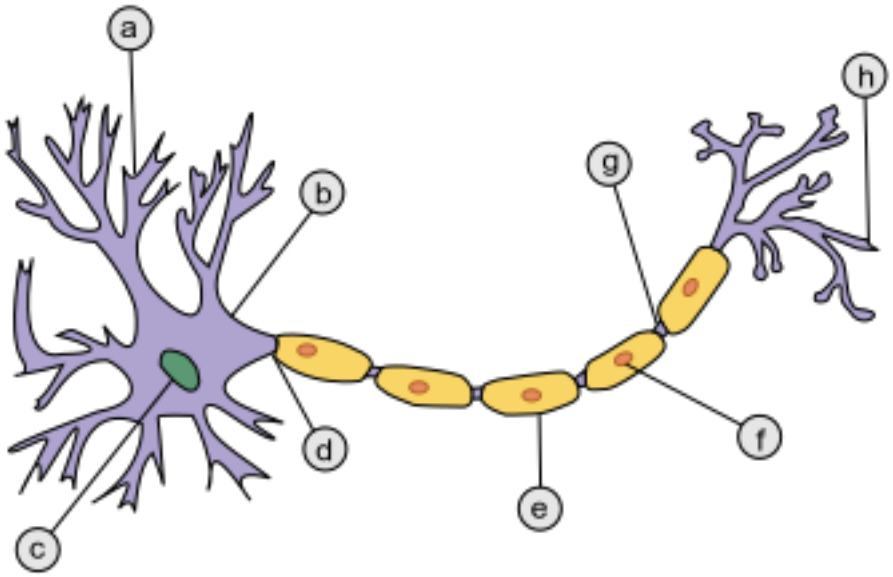

Figure 2.3: A: Dendrite B: Cell body C: Nucleus D: Axon E: Myelin sheath F: Schwann cell G: Node of Ranvier H: Axon terminal [1].

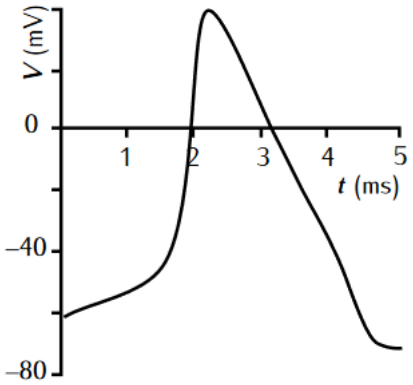

Figure 2.4: The characteristic shape of an action potential seen in the squid giant axon [43].

The Hodgkin-Huxley model is based on squids since they have abnormally large axons which makes observations easier. The neurons of the squid typically has a resting membrane potential of approximately -60 mV, Whenever action potentials arrive at the neuron, there is a small depolarization of the membrane potential. The characteristic shape of the action potentials in squids can be seen in figure 2.4, it has a sharp depolarizing slope at the onset of the spike, with a more moderate re-polarizing phase after reaching around 40 mV. It is due to the different responses each ion channel has to changes in membrane potential that gives the characteristic shape of the action potential.

During the re-polarization phase, it is typical to observe an overshoot,

often referred to as hyper-polarization, where the membrane potential dips below the resting potential of the neuron, consequently making it less susceptible to emitting a new action potential until the membrane potential has re-stabilized. This re-stabilizing phase is referred to as the refractory period.

In contrast to signals in simple conductive transmission lines, an action potential does not attenuate with distance as the mechanisms with which the signals propagate is a regenerative process. There are primarily two types of ions that are vital for the signal propagation and the changes in membrane potential, the sodium ion (Na^+) and potassium ion (K^+). The neurons, axons and dendrites contains these ions in varying concentrations and their relative proportions determine the membrane potential. During resting potential of the neuron, there is a higher concentration of sodium ions in the extracellular space while there is a higher concentration of potassium in the intracellular space. The membrane potential is as the name suggests, the potential measured across the membrane of the cell, and the negatively charged rest potential is due to the more positively charged extracellular fluid compared to the cytoplasm of the cell.

The surface of the different neuronal components consists of a selectively permeable membrane that allows for both diffusion and electric drift. These two forces counteract each other to create the nonzero potential observed in quiescent neurons. In general the cell membrane is considered to consist of three primary components, seen in figure 2.5 that enable the neuron, axon and dendrites to manipulate their polarization. The two first components are the two types of ion channels which are voltage sensitive gates that allow Na^+ to flow into the intracellular space and K^+ to flow out to the extracellular space granted that the voltage requirements are met. The last component is the $Na^+ - K^+$ pump which exchanges $3Na^+$ for $2K^+$ in order to retain the voltage equilibrium of the neuron.

During an action potential the voltage gated ion channels of the cell membrane that are permeable to the sodium ions opens causing a rapid influx of these ions into the cell consequently depolarizing the membrane potential as seen in figure 2.4. When the membrane potential reaches around 30mV, the potassium gates are activated and the membrane is repolarized by allowing K^+ to flow out of the intracellular space. The same mechanisms govern the propagation of action potentials through the axon and dendrites by changing the voltage potentials along the membrane structures.

Hodgkin and Huxley [16] were able to deduce mathematical models for the different ion channels by observing the amount of current carried by the different ions within a squid giant axon by changing the extracellular concentration of sodium. Together these mathematical models can form a realistic simulation of the membranes of neurons, axons and dendrites, although there are many simplifications. The Hodgkin-Huxley model (HH) considers three separate ion currents, namely the aforementioned sodium I_{Na} and potassium currents I_K in addition to a leak current I_L that mainly models the current of chloride ions. In addition to the ionic currents there is a capacitive current I_c which is determined by the capacitance C_m and

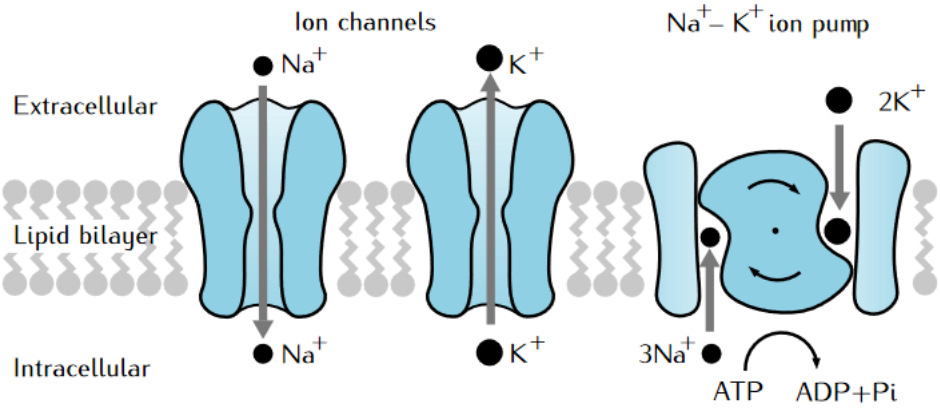

Figure 2.5: The three main components of the neuronal cell membrane. From left to right; Na^+ channel, K^+ channel and a Na^+-K^+ pump that exchanges $3Na^+$ for $2K^+$ [43].

rate of change of potential in relation to time $\frac{dV}{dt}$ for the membrane. These four currents sum up and contribute to the total membrane current given by:

$$I = I_c + I_i = C_m \frac{dV}{dt} + I_i \tag{2.3}$$

where the total ionic current I_i is given by:

$$I_i = I_{Na} + I_K + I_L = g_{Na}(V - E_{Na}) + g_K(V - E_K) + \overline{g}_L(V - E_L) \tag{2.4}$$

where g_{Na}, g_K and $\overline{g}_L$ are the conductances for sodium, potassium and leak respectively. The leak conductance is unchanging, represented as a regular resistor in 2.6, in contrast to the sodium and potassium conductances which are dependent on the membrane potential V and represented as variable resistors in 2.6. The current for each ion channel is not dependent on the actual membrane potential itself, but rather the difference between the membrane potential and the equilibrium potential of that ion channel, referred to as the driving force of the ion. The envisioned electrical circuit that models the four currents can be seen in figure 2.6. The modelling of ion channels based on their voltage-dependent conductance and reversal potential can be used to model different types of neurons as well.

Although it can model a vast range of behaviors, because of its complexity the model is only used to study single or small groups of neurons and is not suited for large scale networks. For the purpose of this project, the Hodgkin-Huxley model is therefor ill-suited since a larger network is required and computational time is a relevant factor. It is however important to understand the underlying biological mechanisms with which the neurons function in order to understand the more simplistic neuron models used in this project.

The Morris-Lecar model [31] has strong similarities to the Hodgkin-Huxley model (HH) with its strong biological plausibility due to the

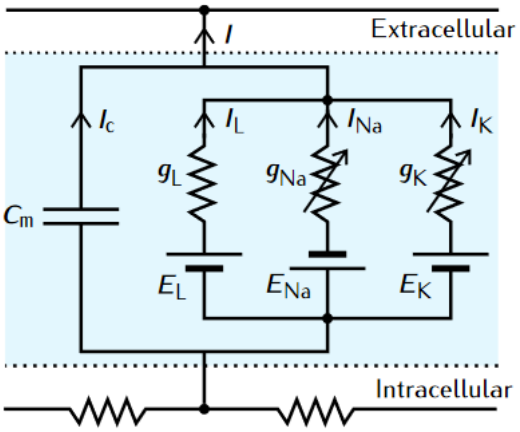

Figure 2.6: The electrical circuit representing the membrane of the squid giant axon [43].

detailed simulation of the mechanisms of the neuron. Similar to the Hodgkin-Huxley model, it is not suited for large scale networks.

2.2.2 Integrate and fire

The neuron model that is most popular among computer scientists and AI researchers is the Leaky Integrate and Fire (LIF), which is a variation of the Integrate and Fire (IF) model. Both the IF and LIF model are based on a circuit model, similarly to the Hudgkin-Huxley model, although a simplified one. IF integrates the incoming current and fires when exceeding a threshold. The LIF model integrates the incoming current in a similar fashion but also includes a leak-constant that models the slow decline of membrane potential when the neuron is in a quiescent state. The equation describing the LIF model can be seen in eq. 2.5.

$$\frac{dv}{dt} = I + a - bv, \text{ if } v \geq v_{thresh}, \text{ then } v \leftarrow c \tag{2.5}$$

v is the membrane potential and I is the sum of incoming current. a, b and c are constants that determine the behavioral dynamics and v_{thresh} is the threshold value that must be exceeded for the neuron to fire a spike. The LIF model is extremely efficient, however this efficiency comes at the cost of simplicity and limits its behavioral repertoire to tonic spikes of constant frequency in response to simple pulses of DC current. It is therefore not suited for neuroscientific simulations where complex neuronal behavior is needed.

There have been a number of subsequent versions of the IF model to address some of its shortcomings like the Integrate-and-Fire-or-Burst [41] that models thalamo-cortical neurons with the inclusion of bursting behavior not present in neither IF nor LIF. The Resonate-and-fire [20] model is able to exhibit sub-threshold oscillations as well as acting as a resonator among other behaviors. Perhaps the most interesting of the IF variations

is the Quadratic-Integrate-and-Fire [26] as it includes spike latencies and a variable threshold.

The FitzHugh-Nagumo model [10] is another model that is fairly efficient but at the cost of biological plausibility. This model can approximate the dynamics of resonator neurons, however it does not exhibit bursting behavior. Another drawback is the requirement to simulate the curve of the action potential reducing the maximum length of each time step consequently reducing computational efficiency. The equations that describe the FitzHugh-Nagumo model can be seen in eq. 2.6 and eq. 2.7.

$$\frac{dv}{dt} = a + bv + cv^2 + dv^3 - u \tag{2.6}$$

$$\frac{du}{dv} = \epsilon(cv - u) \tag{2.7}$$

The change of v per time step is given by 2.6 and includes the constants a, b, c and d constituting a third-degree polynomial subtracted by the variable u, which is updated with 2.7. As mentioned $dt \neq 1ms$, and the tuning of constants is critical for acquiring correct behavior.

The Wilson Polynomial Neurons [50] are described using a set of polynomials and are able to approximate a wide range of neuronal behaviors, however similarly to the FitzHugh-Nagumo model, there is a limit on the time step size which leads to more computations per millisecond putting the Wilson Polynomial Neurons among the less efficient models explored in this work, although it is significantly more efficient than both the Hodgkin-Huxley [16] and Morris-Lecar models [31].

The FitzHugh-Nagumo model, Wilson Polynomial and some of the variations of the integrate-and-fire models could be implemented in relatively large scale networks and exhibit a range of behaviors but they are inferior models for this project when compared to the Izhikevich model, both in term of computational efficiency and in behavioral repertoire.

2.2.3 Izhikevich

The Izhikevich model [18] is an extremely efficient model that mimics the empirically observed behavior of several neuron types. The efficiency of the model allows it to be implemented at large scales. Izhikevich [19] categorize the neuronal spiking types based on observed behavioral characteristics and describe four main classes. The first, *Tonic Spiking*, seen in figure 2.7 (A), observed in Regular Spiking (RS) excitatory neurons, Low-Threshold Spiking (LTS) and Fast Spiking (FS) inhibitory neurons and is characterized by a continuous output of spikes when a continuous current of input is applied. *Phasic Spiking*, seen in figure 2.7 (B), however, fires only a single spike at the beginning of an applied input current. *Tonic Bursting*, seen in figure 2.7 (C), outputs periodic bursts while *Phasic Bursting*, seen in figure 2.7 (D) output a burst at the onset of the input current.

The membrane potential of the neuron is updated with a simple second

order polynomial shown in equation 2.8.

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \tag{2.8}$$

$\frac{dv}{dt}$ is the change in membrane potential in relation to time, v is the current membrane potential, I is the sum of incoming current from pre-synaptic neurons and u is a membrane recovery variable that models the flux of K^+ and Na^+ ions and the change in u in relation to time is given by:

$$\frac{du}{dt} = a(bv - u) \tag{2.9}$$

Membrane voltage is in mV and time is given as milliseconds. To model the rapid depolarization in membrane potential following a spike, v is updated with 2.10.

$$\text{if } v \geq 30, \text{ then } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{2.10}$$

The values of the constants a, b, c and d determine the type of neurons, which includes regular spiking (RS), intrinsically bursting (IB), chattering (CH), fast spiking (FS), thalamo-cortical (TC), resonator (RZ) and low-theshold spiking (LTS) neurons. These different neuron models portray the plethora of diverse neurons found in the brain and exhibit different behaviors.

Since a great focus of this project is on taking inspiration from the biological brain, the biological plausibility of the Izhikevich model in addition to its computational efficiency makes it the best model, made evident by 2.2.

2.3 Learning Methods

In general the learning methods for machine learning models can be divided into two classes, namely supervised learning and unsupervised learning methods. The way in which the human brain learns is not fully understood, but researchers have proposed a plethora of methods that fall under both of these classes with a varying degree of realism.

This section will present some current learning methods used in machine learning models in addition to the mechanisms of biological learning. The reason for presenting relevant aspects of biological learning is to highlight the lack of biological plausibility for some of the existing learning methods and demonstrate the mechanisms that inspired the proposed learning method.

2.3.1 Biological learning

Similarly to neural networks, the learning that takes place in biological brains is related to changes in connection strength and on larger time scales, even the creation or destruction of connections. These changes

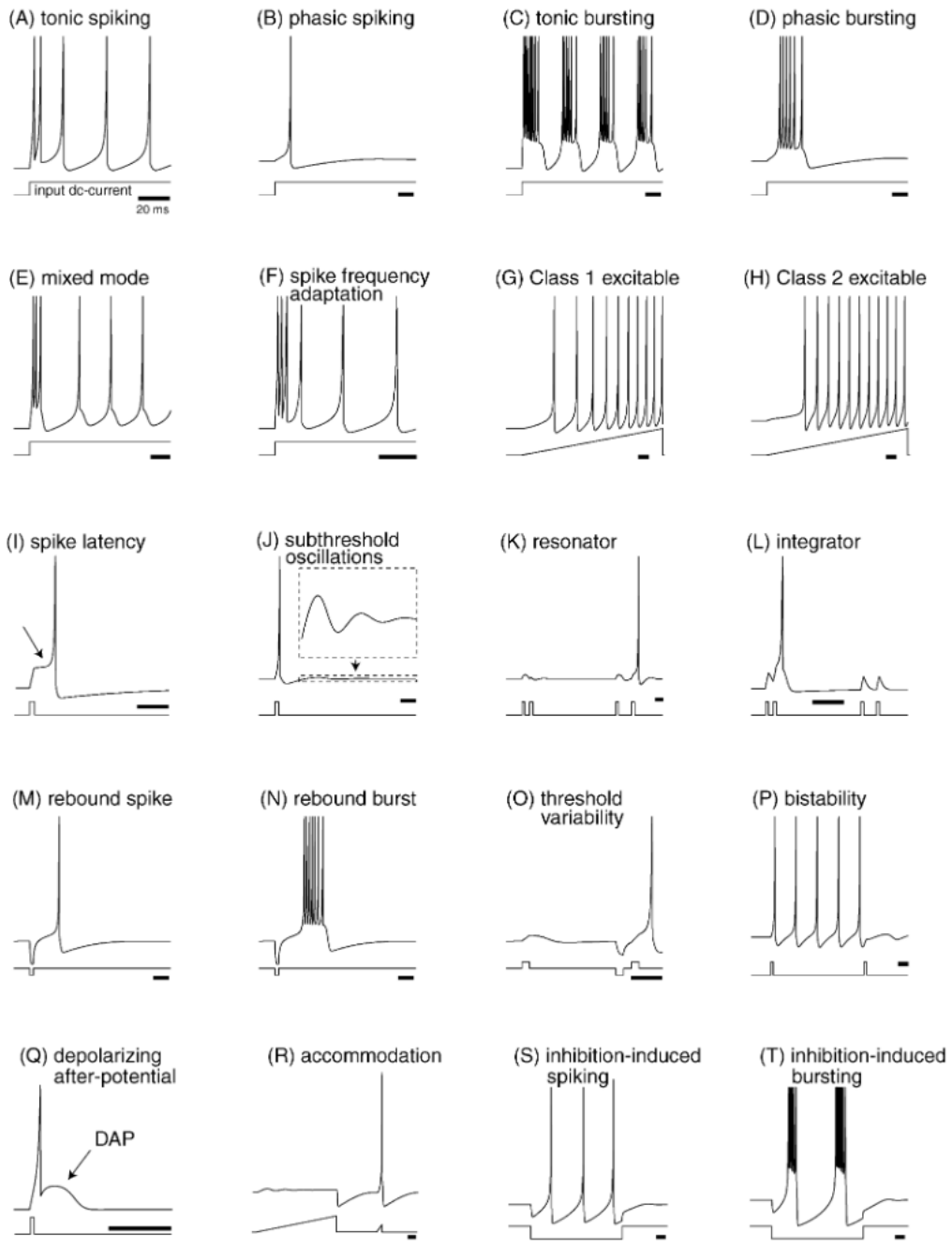

Figure 2.7: Shows the neuronal behaviors that is exhibited by tweaking the parameters of the Izhikevich model [19]

in synaptic strength is due to the plasticity of synapses, which allows them to continually adapt to local activity. The adaptation can be on smaller time scales referred to as short-term plasticity (STP) or can be more permanent referred to as long-term plasticity (LTP). Long-term plasticity refers to functional changes that equates to memory and learning. Short-term plasticity works on the millisecond time scale, and a common

example of STP is short-term synaptic depression (STSD) or short-term synaptic facilitation (STSF). These two processes respectively decrease and increase the effect a pre-synaptic spike has on a post-synaptic neuron but only temporarily. If an action potential causes STSD, the following action potential will have a lessened effect on the post synaptic neuron, consequently the significance of the first action potential is greater than the second action potential and vice-versa if the action potential causes STSF [32].

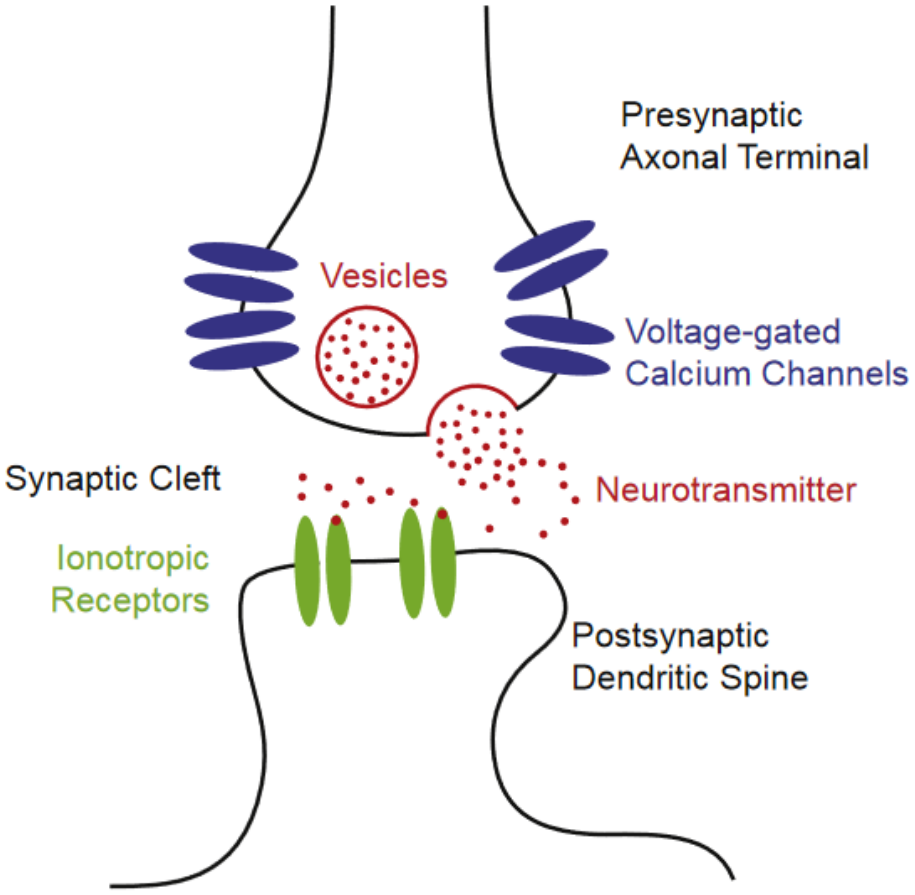

Figure 2.8: The signal transmission across the synaptic cleft is facilitated by the release of neurotransmitters at the presynaptic terminal which are received by the receptors at the postsynaptic terminal [32].

The signal transmission across the synaptic cleft is facilitated by the release of neurotransmitters at the pre-synaptic terminal which are received by the receptors at the postsynaptic terminal as seen in figure 2.8. The release of neurotransmitters occurs when an action potential activates the voltage sensitive calcium channels at the post synaptic terminal resulting in an influx of calcium ions which triggers the release of the neurotransmitters.

The biological mechanism that causes STSD is the reduction of neurotransmitters released at the pre-synaptic terminal. It has been theorized

that STSF is caused by residual calcium at the pre synaptic terminal, however it has not been confirmed [32].

The communication between neurons can be excitatory or inhibitory and the main deciding factor is the type of neurotransmitter released. The most common inhibitory neurotransmitter in mammals is GABA, but it is the group of corresponding receptors that determine the mechanism with which the inhibition occurs. The receptor that causes the greatest inhibitory effect is the $GABA_A$ receptor, which is done by reducing the overall membrane resistance lessening the voltage change caused by incoming excitatory signals. The most common excitatory neurotransmitter is glutamate.

Taking inspiration from the biological learning mechanisms of the brain, the proposed learning method in this project mediates delays based on local activity.

2.3.2 Local Learning

Local learning is the group of learning methods that relate to changes that occur based on local information or activity. This type of learning is believed to be the primary mechanism for learning in the brain.

A well-known and documented local learning mechanism is Spike Time Dependent plasticity (STDP). In its simplest form STDP is, as its name suggests, a mechanism where the synaptic strength is altered based on the relative timing of spikes at pre- and post-synaptic neurons. The idea is that if there is a causal relationship between pre- and post-synaptic activity, then this effect should be increased proportionally to the strength of the causality, whether it be inhibitory or excitatory.

Song and Miller [42] formulates an STDP rule according to equation 2.11,

$$F(\Delta t) = \begin{cases} A_+ e^{\Delta t/\tau_+}, & \text{if } \Delta t < 0 \\ -A_- e^{-\Delta t/\tau_-}, & \text{if } \Delta t > 0 \end{cases} \tag{2.11}$$

where A_+ and A_- are the maximum axonal modification as Δt goes towards 0, while tau_+ and tau_- determines the pre- and post-spike time window where axonal change occurs. Δt is given by $\Delta t = (t_{pre} + d_{prepost}) - t_{post}$, where t_{pre} and t_{post} is the spike time of the pre- and post-synaptic neuron respectively. The output of $F(\Delta t)$ does not give the absolute change in weight, but rather the change relative to the maximum weight value $\overline{w}_{max}$. Thus, the change in weight is given by $\Delta w = F(\Delta t) \cdot \overline{w}_{max}$. Figure 2.9 shows the plotted function with maximum relative change set to $A_+ = A_- \approx 0.5$. The absolute value of the derivative of each section of the discontinuous function is larger closer to the y-axis leading to a more dramatic change to the weight as the pre-synaptic spike gets closer to the post-synaptic spike time. As the spike times diverge, the change to the weight lessens. The assumption is that the closer the pre-synaptic spike is to the post-synaptic spike time, while still arriving before it, the stronger the causal relationship is and consequently more potentiation should be applied to the synapse. Inversely, the closer the pre-synaptic spike is to the

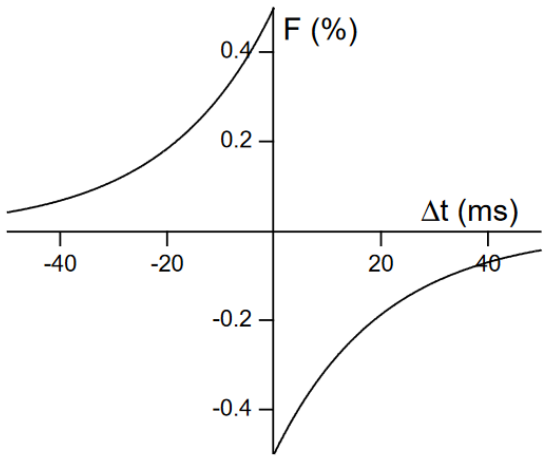

Figure 2.9: The percentage change relative to the maximum allowed weight can be seen along the y-axis as a response to Δt [42].

post-synaptic spike time, while still arriving after it, the weaker the causal relationship is, and more depression should be applied to the synapse. This learning method is simple yet powerful, and allows the network to strengthen the paths that correlates to similar inputs, thus the network is able to group the input space into a number of classes.

In the context of delayed connections, the STDP mechanism just described, played an important role as inspiration for the proposed learning method which mediate the delays instead of weights based on the local activity.

2.3.3 Supervised Learning

Supervised learning is a method commonly used in conventional ANNs and is a big contributing factor to the success of modern machine learning. The method requires a labeled data set, often thousands of samples, in order to train the model. The training is done by giving some input data to the model, the model predicts an output and the output is compared to the label and the resulting discrepancy is used to modify the model so that the error is reduced. This process is repeated until the model performs satisfactorily. The discrepancy is calculated using a loss function. A typical loss function is the mean squared error 2.12,

$$J(W) = \frac{1}{n}\sum_{i=1}^{n}(h_W(x^{(i)}) - y_i)^2 \tag{2.12}$$

where y_i is the label value for training data i and $h_w(x^{(i)})$ is the hypothesis function or predicted value which can also be denoted as $\hat{y}_i$. The purpose of the training is to minimize the loss function $\min_w J(w)$, and this is commonly done through gradient descent. Since the model consists of a set of weights which determine the output, the goal is to find the combination of weights

that give the least error overall, and this is done by calculating the gradient of the error function and changing the parameters accordingly. The update to a weight w_i is done by subtracting the partial derivative of the loss function with respect to w_i multiplied by a learning rate a that determine the magnitude of the change. This can be written as 2.13.

$$w_i = w_i - \alpha \frac{\partial J(w_0, w_1, w_2, \ldots, w_n)}{\partial w_i} \tag{2.13}$$

The process can be visualized as seeking the lowest point of a mountainous terrain 2.10, although the parameter space for any deep neural network would be high dimensional and impossible to actually visualize, but the principle still holds.

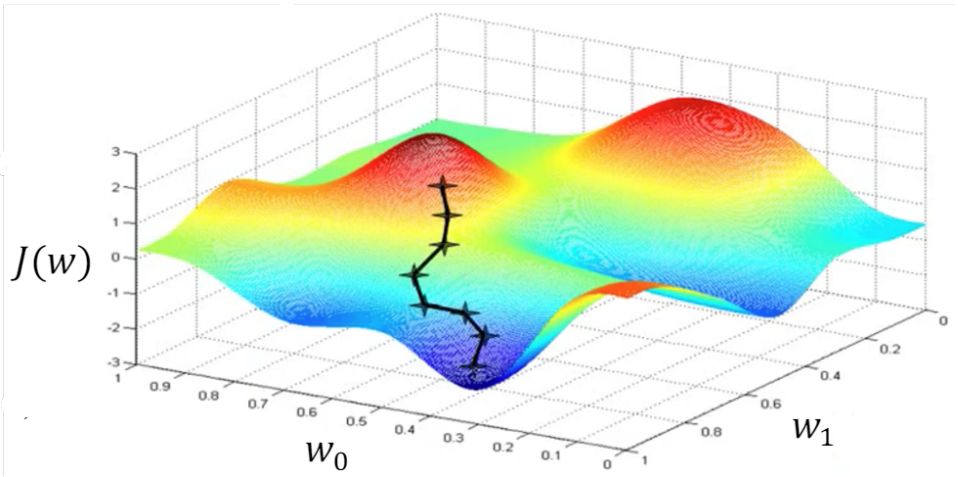

Figure 2.10: The parameter space for a model with two weights, w_0 and w_1. The algorithm iteratively descends towards the optimal weight values where $J(W)$ is at its minimum.

But the partial derivative becomes a bit more complicated in DNNs. As previously mentioned, conventional feed-forward DNNs consist of a set of n weighted connections $W = \{w_0, w_1, w_2 \ldots, w_n\}$, configured in layers with activation functions and the input is propagated through the layers until the output layer where the prediction is given. Since the output of each layer is the result of the output of all previous layers, the process of calculating error and adjusting weights must therefore be done by propagating the error backwards through each layer. This method for updating the weights of a neural network is called error backpropagation (EBP). Since the layers are dependent on each other, chain derivatives must be used to uncover the correlation between changes in the weights and changes in the cost function. If presented with a classification problem and the cost function is given by 2.14,

$$C_k = \sum_{j=0}^{n_L - 1} (a_j^{(L)} - y_j)^2 \tag{2.14}$$

where k indicates the training sample and $a_j^{(L)}$ is the activation or output of the j'th neuron of the L'th layer and given by 2.15

$$a_j^{(L)} = g(z_j^{(L)}) \tag{2.15}$$

where g is some nonlinear activation function and $z_j^{(L)}$ is the weighted sum given by 2.16,

$$z_j^{(L)} = \sum_{k=0}^{n-1} w_{jk}^{(L)} a_k^{(L-1)} + b^L \tag{2.16}$$

where $w_{jk}^{(L)}$ is the weighted connection between neuron k in layer $L-1$ and neuron j in layer L, then changes in C with respect to changes in $w_{jk}^{(L)}$ is given by the chain derivative seen in eq. 2.17.

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}} \tag{2.17}$$

This is the gradient of the cost function with respect to that particular weight for one training example. The gradient for the other weights can be found in a similar fashion by finding the chain derivatives that describe the relationship between that weight and the cost function. The training samples are often split into batches and and the gradient components are therefore averaged over the batch. DNN also includes biases but the process is the same for calculating their gradients.

The process of training ANNs with EBP is very powerful and has resulted in models that far exceeds human capabilities on certain tasks. In a biological context, the existence of backpropagation is highly debated. Since EBP requires error information from downstream neurons in order to update connections warranting some bidirectional signal propagation, and synaptic plasticity is believed to be the result of local activity only, it is unlikely that it exists in the brain. Although there are bidirectional connections in the brain, they are not always present, and EBP would require identical strength for both connections requiring some mechanism for synchronizing them which has not been observed.

EBP works extremely well in conventional ANNs, however when trying to implementing EBP in SNNs one is faced with the difficult task of of finding the derivative of the neuronal output [49].

Similarly to supervised learning in conventional ANNs, the goal is to match the model output to the desired output and in the case of SNNs the process of supervised learning is to train the model to reproduce the desired spatiotemporal pattern of spikes. A spatiotemporal pattern of spikes, or a spike train, can be mathematically described by 2.18

$$s(t) = \sum_{f=1}^{F} \delta(t - t^f) \tag{2.18}$$

t^f is the spike at position f in the ordered sequence of spikes and the function $s(t)$ gives the number of spikes for time t as the Dirac delta

function is given by 2.19 resulting in only spikes that occur at time t to be counted.

$$\delta(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x \neq 0 \end{cases} \tag{2.19}$$

Similar to conventional ANNs, the output S_o of the model is determined by the input, which is a spike train S_i in the case of SNN, and the weights of the model W. This creates a functional relationship given by 2.20.

$$S_o = F(S_i, W) \tag{2.20}$$

And similar to conventional ANNs the purpose of the training is to minimize the error function which can be described by equation 2.21. This function compares the output spike train $S_o^m(t)$ for each output neuron m with the desired spike train $S_o^m(t)$ that corresponds to the given output neuron. These comparisons are summed and a set of weights W should be chosen such that the summation is at its minimum value [48].

$$min(E(S_o, S_d)) = min \sum_{m=1}^{N_o} |s_d^m(t) - s_o^m(t)| \tag{2.21}$$

This learning method is based on strengthening connections where spikes are desired and weakening connections where spikes are not desired.

Bohté, Kok and Pouté [3] proposed a method called *SpikeProp* for implementing a learning mechanism similar to EBP in networks of spiking neurons. The connections between neurons consist of a constant m number of sub-connections each with its own weight and delay as seen in figure 2.11. The weight of connection k from neuron i to neuron j is denoted as w_{ij}^k. The delay for connection k is denoted as d^k. A spike from neuron i at time t is denoted as $y_i^k(t)$. The input for post-synaptic neuron j is therefore given by the sum of each neurons weighted sub-connections k over the sum of all connected neurons i to m as seen in eq. 2.22.

$$x_j(t) = \sum_i^n \sum_k^m w_{ij}^k y_i^k(t) \tag{2.22}$$

The supervised aspect of SpikeProp is done by comparing the models spike-train output t_j to a desired spike-train t_j^d using the least mean squared error function. Just like for EBP in conventional ANNs, chain derivatives for the correlation between error function and connection weights must be calculated 2.23.

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial t_j}\frac{\partial t_j}{\partial w_{ij}^k} = \frac{\partial E}{\partial t_j}\frac{\partial t_J}{\partial x_j(t)}|_{t=t_j}\frac{\partial x_j(t)}{\partial w_{ij}^k}|_{t=t_j} \tag{2.23}$$

The partial derivative of the output spike-train in relation to the corresponding weight is expanded to include the partial derivative of the spike-train with respect to the post-synaptic input and the partial derivative of the post-synaptic input with respect to the synaptic weights. These partial

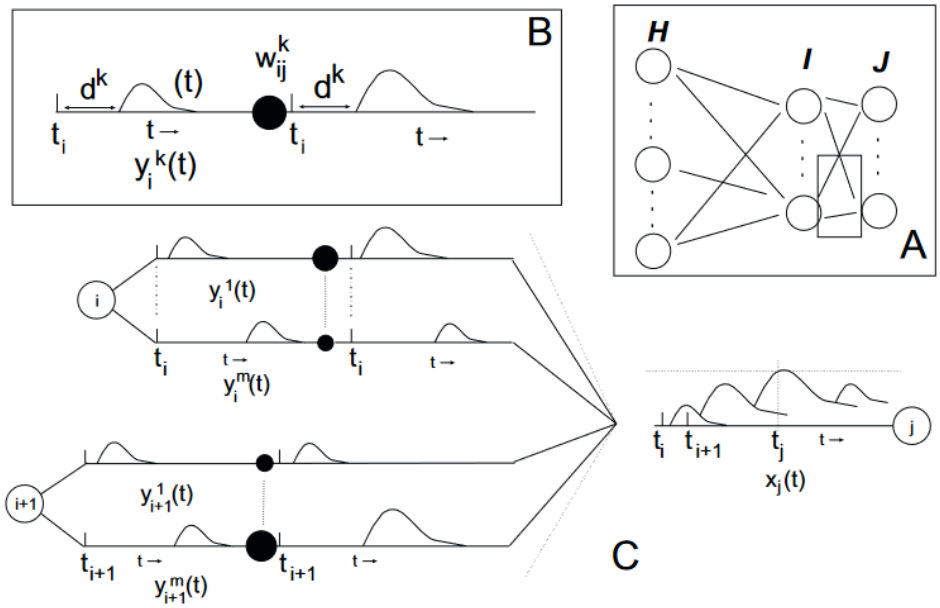

Figure 2.11: A: shows a feed-forward SNN with input layer denoted as H, the hidden layer denoted as I and the output layer denoted as J. B: shows the connection k from neuron i to neuron j weighted by w_{ij}^k, the delay is denoted as d^k and the spike-train is given as a function of time t and denoted as $y_i^k(t)$. C: show two pre-synaptic neurons with multiple delayed connections. The input is summed at the post-synaptic neuron and membrane potential is given by $x_j(t)$ [3].

derivatives along with the learning rate is used to update the weights similar to regular gradient descent. The issue, as stated earlier, is the discontinuity related to spiking neurons, and SpikeProp overcomes this obstacle by approximating $x_j(t)$ as a linear function of time. This method was later improved by Schrauwen and Van Campenhout [40]. However, these methods updates the weights and not the delays of the network. They are also very unlikely to occur in biological neural networks.

2.4 Coding Schemes

Neurons communicate with each other through APs, and although a single AP can contain a lot of information by itself, for simplicity's sake we can consider it as a boolean value; either there is an AP or there is not. When combining these individual spikes with precise spike timings into series of consecutive spikes, or combinations of spikes from different neurons, there is a significant number of ways that information can be encoded. These coding schemes vary in biological plausibility and their practicality in terms of implementation in SNNs relies heavily on the network configuration, input type and how output is extracted from the network. The most common, or otherwise relevant, coding schemes are

presented in this section.

2.4.1 Rate Coding

The coding scheme that was initially believed to apply for biological neural networks was Rate Coding. This coding scheme pertains, as the name suggests, to the rate at which a neuron fires and was believed to be the sole encoder of information. In this coding scheme all information is contained within the firing rate and the firing rate is simply the number of spikes divided by a temporal sampling window. Although, it is now common knowledge that the brain employs various coding schemes, one area of the brain that is still believed to implement rate coding is the areas responsible for muscle control, where a higher firing rate equates to a stronger muscle contraction [9].

Since the inclusion of a certain sampling window is an intrinsic feature of rate coding, the speed at which a signal may propagate through a network is consequently limited by this sampling window. However, spike-rate encoding is considered quite robust and applicable to many tasks. A practical example of how information can be encoded with this method is to convert image pixels to corresponding spike-rates based on pixel intensities. From the viewpoint of computational neuroscience, other coding schemes might be more suitable for model implementation due to power and computational efficiency. However, since this is one of the most common encoding method in SNNs, it was used as a comparison for other methods.

2.4.2 Rank Order Coding

In the work of Thorpe and Gautrais [46], a coding scheme coined Rank Order Coding (ROC) is hypothesised where the information is encoded as the order in which a set of neurons spike.

When considering a set of neurons N, in terms of order, at which ROC is dependent, the number of possibilities is given by $N!$. This gives $log_2(N!)$ bits of information. For $N = 6$, ROC can encode around 9 bits of information for a time window of 6ms allowing for one spike per ms, in other words a perfect scenario for the selected temporal resolution.

Figure 2.12 illustrates how the order of pre-synaptic spikes can affect the post-synaptic response. In this example, the post-synaptic neuron is connected to sixteen pre-synaptic neurons with various connections strengths. If the order of incoming spikes matches the order of weighted connections from strongest to weakest, the activation of the post-synaptic neuron is maximized. If the order is reversed, the activation is minimized. However, this specific implementation relies on a decreasing sensitivity to inputs in the post-synaptic neuron, which means altering the Izhikevihch model, something that was not desired.

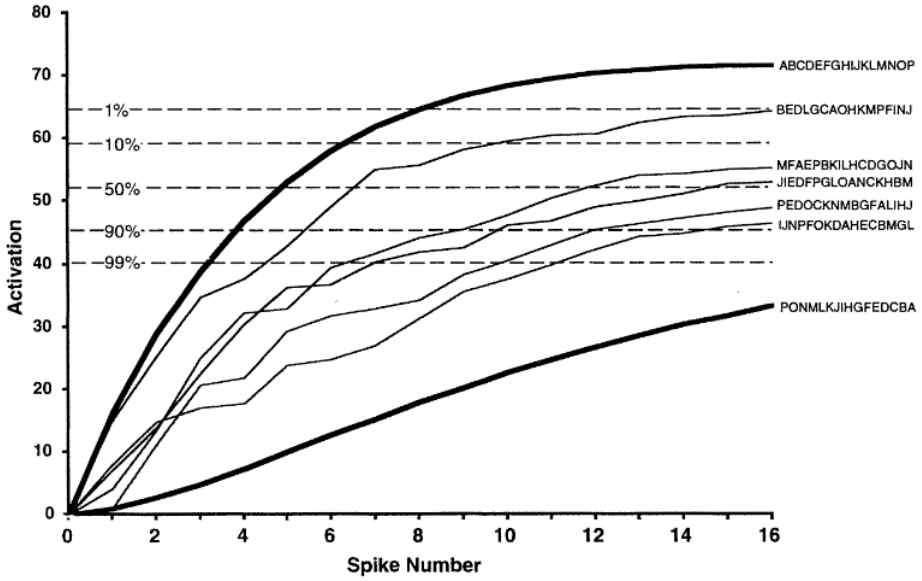

Figure 2.12: The activation of the post-synaptic neuron is dependent on the order of incoming signals. When the order of incoming signals match the order of weights from strongest to weakest, the activation is maximized. If the order is reversed the activation is minimized [46].

2.4.3 Relative spike latencies

Although the ROC scheme is simple and elegant, there is some degree of information loss as the method only considers the spike order, and not the intervals between the spikes. If the ROC method is compared to a coding scheme that is contingent on the relative spike-timings of a group of neurons, when applied to $N = 6$ neurons and a time window of $T = 6$ ms, the latter method can encode $log_2(T^N)$, or 15 bits of information. This is significantly more than the 9 bits encoded by the ROC method.

Gollisch and Meister [12] proposed a method where information is encoded in the relative spike latencies (RSL) of a set of neurons. They argue that this might be the method that is employed in the visual center of biological brains. This method not only differentiates the spikes based on order, but also allows for encoding the relative difference of each input in their relative latencies.

Due to its powerful encoding capabilities, and relatively easy implementation, this method was extensively used in this project.

2.4.4 Time-to-first-spike

Time-to-first-spike is a coding scheme that is believed to be responsible for encoding information in areas of the brain that requires fast responses. Park et al. [33] proposed a method using a decaying threshold for the input neurons such that inputs with higher values would result in action potentials at an earlier stage then weaker inputs. This threshold is given by

equation 2.24, where θ_0 is a threshold constant and τ_{th} is a time constant. Once a spike is emitted from an input neuron, the neuron can no longer fire, in other words, each neuron only fires ones.

$$P_{th}(t) = \theta_0 e^{\frac{-t}{\tau_{th}}} \tag{2.24}$$

In the decoding phase the arrival of the spike determines its significance in a similar way as ROC. However, in contrast to ROC, the effect the action potential has on the post-synaptic neuron decreases with time and not with the order at which the spike arrives. The weighted sum $z_j(t)$ that is the input for the post-synaptic neuron j at time t is therefore mediated by a time-decaying component $w_s(t)$ and given by 2.25.

$$z_j(t) = w_s(t) \sum_i w_{ij} s_i(t) \tag{2.25}$$

The time decaying component is given by 2.26.

$$w_s(t) = e^{\frac{-t}{\tau_s}} \tag{2.26}$$

The time-to-first-spike method is very energy efficient as it limits the amount of spikes required to transmit information. This method resonates well with a learning method that decreases delays between causally related spikes, and perhaps increases delays between non-causally related spikes. However, the method proposed in this project will not focus om promoting fast throughput for for connections with causally related spikes, and will therefore not be used.

2.4.5 Other coding schemes

There exists several other coding schemes that can be applied to SNNs. Among them is the method proposed by Kim et al. [25], where input data is represented as their binary values. A spike represents a binary bit-value of 1, and the absence of a spike represent a binary bit-value of 0. The different bits are distinguished by splitting the sequence into temporal phases. The phase determines the current weight of the connection and consequently the relative position of the spike can be determined. The time-varying weight is given by 2.27.

$$w_s(t) = 2^{-(1+mod(t-1,8))} \tag{2.27}$$

The number of phases is determined by the number of bits needed to represent the largest value of the input data. If the input is an image of pixels with 8-bit values, then 8 phases are needed. For the decoding phase, the weight of the spikes is used to determine the significance of the spike, and the higher position the spike is in the binary number the more impact it has on the post-synaptic neuron. This method is not biologically plausible and not practical for this project.

Burst coding is another coding scheme that is believed to exist in the biological brain [51]. This method encodes the input value as a

proportionally intensive burst. A higher input value gives a burst with a higher number of spikes and a lower inter-spike interval. This method is not applied in this project as it requires a lot of spikes and has a strong resemblance to rate-coding.

2.5 Code libraries

In order to simulate delay learning in neural networks, a simulation software must be used. Early in the second phase of this project a process was started to evaluate the software libraries that are available for simulating spiking neurons. In contrast to popular libraries like TensorFlow and Pytorch which allow for the training of conventional neural networks, the slew of libraries that are targeted toward Spiking Neurons are much less known and vary greatly in available documentation and features. The initial step was therefore to identify the key features that was believed to be important for the project at that time. These features included the implementation of delays, allowing for recurrent connectivity, allowing for the adjustment of delays in real-time, including both the Izhikevich- and LIF-model and GPU-support for faster simulation. Other indicators of whether the libraries were viable is how recent the latest update to the code is and my personal evaluation. The entire table can be seen in table 2.1.

Table 2.1: Evaluation of Spiking Neural Network libraries.

Framework	Delays	Reccurrent connectivity	Learning method	Izhikevich	LIF	GPU support	Latest update	Opinion	Sum
ANNarchy	3	3	2	3	3	1	2021	1	0.5
NEST	3	2	2	3	3	1	2021	2	0.6
cuSNN	1	1	1	1	3	3	2018	1	0.4
CarlSim	1	1	1	3	3	3	2021	1	0.5
NeMo	3	3	2	3	3	3	2015	4	0.7
PyNN	2	3	2	2	3	3	2021	2	0.6
Brian2	3	3	2	3	3	1	2020	4	0.8
BindsNET	1	1	2	2	3	3	2019	1	0.4

There were eight libraries that seemed relevant to explore further. After reading through the documentation of each, a score was given from 1 to 3 for each of the key features, with 1 being that the feature is missing, 2 being unresolved and 3 meaning that the feature is present. My own opinion was scored from 1 to 5 as I noticed a clear difference between the viability and usability of the different libraries and three categories was insufficient to differentiate between them. The latest update to the code was scored such that the oldest update was given the lowest score, and the latest update was given the highest score.

Based on the scores calculated, the Brian2 library was investigated further. Initially it appeared that the library implemented the most important features. The library allows for specifying neuron models by way of differential equations. It also includes axonal delays; however, it is not made for delay learning as the mechanisms for updating connection weights can not be utilized properly for delay changes. Eventually I found a way to have real-time delay updates, but this method drastically reduced simulation efficiency, and the library also exhibited numerical instability

for high-activity networks. PyNN, NeMo and NEST were subsequently tested and also deemed unfit for the project.

As no appropriate library that fit the requirements for this project was found, a new library was created using Python. This code is described further in section 3.1.

2.6 Related Works: Delay learning in Spiking Neural Networks

The focus of this project was to develop a learning mechanism that alters the propagation delays of action potentials as they travel between spiking neurons, and the process of adjusting these delays can be referred to as delay learning. This section will therefore present existing work that either incorporates some form of delay learning or in some way specifically relies on the delays for computation.

Bohté, Kok and Poutré proposed a method named SpikeProp which is an approximation of the error back propagation method used in conventional neural network [3]. Although this method does not explicitly alter the delays, the delays play an important role in computation and are not homogeneous. This implementation is limited to a feed-forward topology, which is remedied in the work of Bellec et al. [2], where error-back-propagation through time is applied to recurrent spiking neural networks. Again, the delays are static, but still plays a role in computation.

Both Schrauwen and Campenhout [39] and Wang, Lin and Dang [47] applies supervised delay learning in the form of error back propagation in their feed-forward SNNs. The latter work converts the spike trains to continuous analog signals allowing for the application of common mathematical operations. The former work extended the original SpikeProp model to accommodate for delay learning, a change that could be considered an improvement of the biological plausibility of the original model. Their main motivation however, was to reduce the size of the network since the original SpikeProp algorithm required multiple delayed synaptic terminals for each connection. Their model still required multiple delayed terminals, but the number was reduced since the delays could be adjusted to accommodate for the removed delayed connections. The proposed model can be seen in figure 2.13 where each connection consists of multiple terminals, each with their own delay d_{ij}^n, weight w_{ij}^n and post synaptic potential ϵ_{ij}^n.

Johnston and Prasad [24] present a hybrid method which includes both unsupervised learning through STDP and supervised learning through a genetic algorithm. However, it is the genetic algorithm that mediates the delays, and therefore the delay learning method can be considered neither online nor local learning.

The model proposed by Taherkhani et al. [44] rely on a version of STDP which incorporates supervised learning. The supervised aspect stems from an instruction signal sent from a neuron in the output layer to a neuron in the hidden layer instructing potentiation or depression of its weights based on the desired output of the output neuron. This change in weights is done

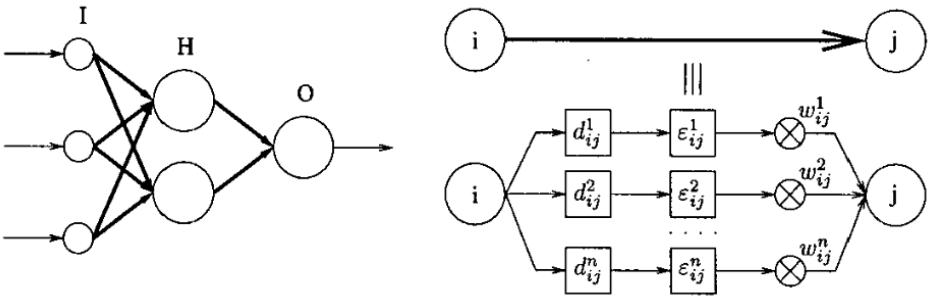

Figure 2.13: An overview of the model used in the delay learning extension of SpikeProp where each connection consists of several sub-connections with their own delay d_{ij}^n, weight w_{ij}^n and post synaptic potential ϵ_{ij}^n [39].

through STDP, in other words, the connections to pre-synaptic neurons that fired in the desired time interval are potentiated. Likewise, connections from inhibitory neurons are depressed based on anti-STDP. The delay of the connection between the hidden neuron and the output neuron is then shifted to allow the spike to arrive at the output neuron in the desired time window. An illustration of the method can be seen in figure 2.14. The top-

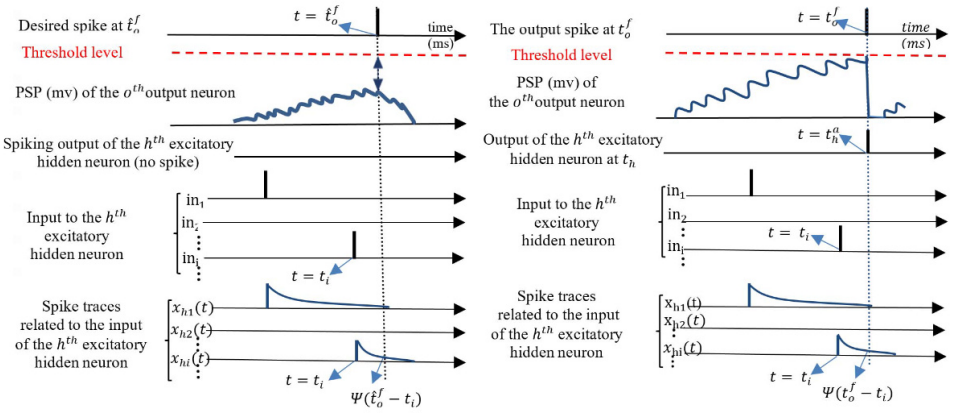

Figure 2.14: An overview of the supervised STDP method where connections are adjusted according to an instruction signal [44].

left plot shows the desired time for the output spike and the inadequate membrane potential of the $o'th$ output neuron. The plots below shows the missing output spike of the $h'th$ hidden neuron. The right plot shows the occurrence of an output spike from the output neuron when the effects of the input neurons to the $h'th$ hidden neuron is potentiated resulting in the generation of an action potential which is passed to the output neuron.

Another supervised algorithm that implements delay learning through an instruction signal is the DL-ReSuMe [45] which combines the weight adjustments of the ReSuMe algorithm [36] with delay shifts. The method used in ReSuMe to adjust weights is a combination of Hebbian and anti-

Hebbian learning, that enabled the model to learn a desired spike train based on a input of a spatiotemporal spike pattern. A similar method is used to adjust delays in the DL-ReSuMe method by checking for desired and undesired spikes. If a spike is desired at a point in time and no spike is present, the connection that provides a spike in the closest temporal proximity to the desired spike time is delayed in order to increase the membrane potential of the post-synaptic neurons, possibly allowing it to exceed its threshold and produce a spike. Similarly if a spike is present without the desire for a spike, the inhibitory connection with a spike in closest temporal proximity to the undesired spike time will be delayed in order to reduce the membrane potential and possibly avoid a spike being produced in the post-synaptic neuron.

Paugam-Moisy, Martinez and Bengio [34] proposes a method that they refer to as *Multi-timescale learning* for its combination of a small time-scale learning method and a large time-scale learning method. Inputs are given to the model in the form of vectors containing numerical values. These values are converted with a temporal coding scheme where higher values results in earlier firing times for the corresponding input neuron. The first output neuron to fire corresponds to the prediction made by the model. Successive inputs are separated by large intervals to avoid interference.

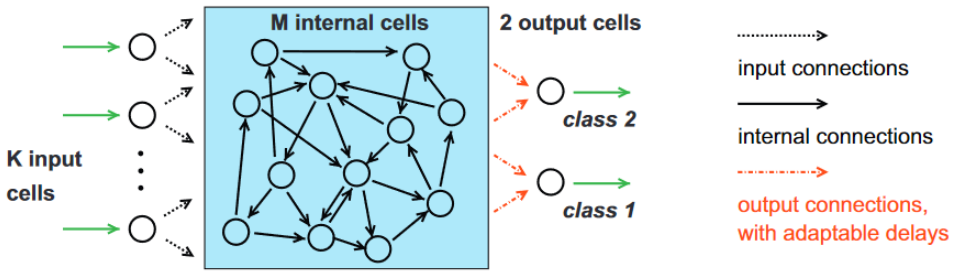

Figure 2.15: Shows the proposed LSM with a set K of input neurons, a set M of internal cells and one output neuron per class. The edges connecting the reservoir to the output layer has adjustable delays [34].

The architecture of the proposed model can be seen in 2.15. The weights of the input and output connections are static and unchanging. The input connections are chosen so that they result in sufficient excitation of the membrane potential of the hidden neurons so they produce spikes. The input neurons are connected to the hidden neurons with a probability P_{in} and with no delay, while the reservoir is fully connected to the output layer with trained delays. Plasticity through STDP is only present in the reservoir and only affects weights, while delays of the reservoir are kept static.

Their proposed combination of learning methods that they implement is STDP for the small time-scale method and a supervised method that adapts the delays that connects the reservoir to the output neurons for the large-scale method. This supervised learning method relies on the existence of polychronous groups (PG) within the reservoir. PGs are groups of neurons that responds in a reliable and reproducible way to specific

input patterns. For a given set of input patterns with corresponding output neurons representing each patterns class, the connections between the reservoir and the output neurons are iteratively adjusted depending on whether the model classified correctly or not. If the model classified correctly, no changes are made, but if it classified incorrectly the delay of one triggering connection to the correct output neuron is decreased. Likewise the delay of a triggering connection to the incorrect neuron that gave the false classification is increased. A triggering connection is connection that provided a spike to the output neuron which resulted in a spike in the output neuron. This iterative process of small delay adjustments gradually increases the performance of the model. To summarize, the STDP mechanism continually adjust reservoir weights, while the supervised learning method adjusts delays once for each input pattern. The article mentions the possibility of including a plasticity rule that adapts the delays of the reservoir neurons, but this was not implemented.

Based on these findings, it does not appear that local learning rules for explicit delay learning has been implemented, substantiating the need for exploring such a mechanism.

Chapter 3

Framework and implementation

This chapter will cover the simulation framework and the details concerning the implementation of transmission delay model. The implementation of the neuron model will also be covered in addition to thorough testing of the model behavior. The chapter will conclude with a demonstration of the model by recreating the polychronous groups described in the 2006 paper by Izhikevich [21].

3.1 Framework overview

Based on the lack of suitable existing frameworks as made clear by the testing described in section 2.5, an entirely new framework was required. Although it is time-consuming to develop such a framework, it comes with many benefits as the framework can be tailor made to a specific purpose, removing unnecessary features.

The programming language used was Python, as it is easy and quick to program, and can be quite efficient if some care is taken with the choice of data structures and algorithmic implementation. The resulting code showed a significant improvement in simulation time over the Brian2 library.

A class diagram of the most important parts of the code can be seen in figure 3.1. The Population class is the main driver of the simulation. It holds both the neurons and the synapses of the network configuration and contains the *run* method which runs the simulation by calling the update functions of all the neuron and synapse objects. The *run* method also contain a call to the method that builds the polychronous groups, which are also stored in the population object.

The population object also holds the methods for creating the neurons as well as the synapses, which can be created individually or through various helper functions that will create specific network topologies such as feed-forward networks, ring lattices or reservoirs. Lastly, the Population class holds the methods for plotting simulation related data, like membrane potentials of the neurons, raster plots or network topologies.

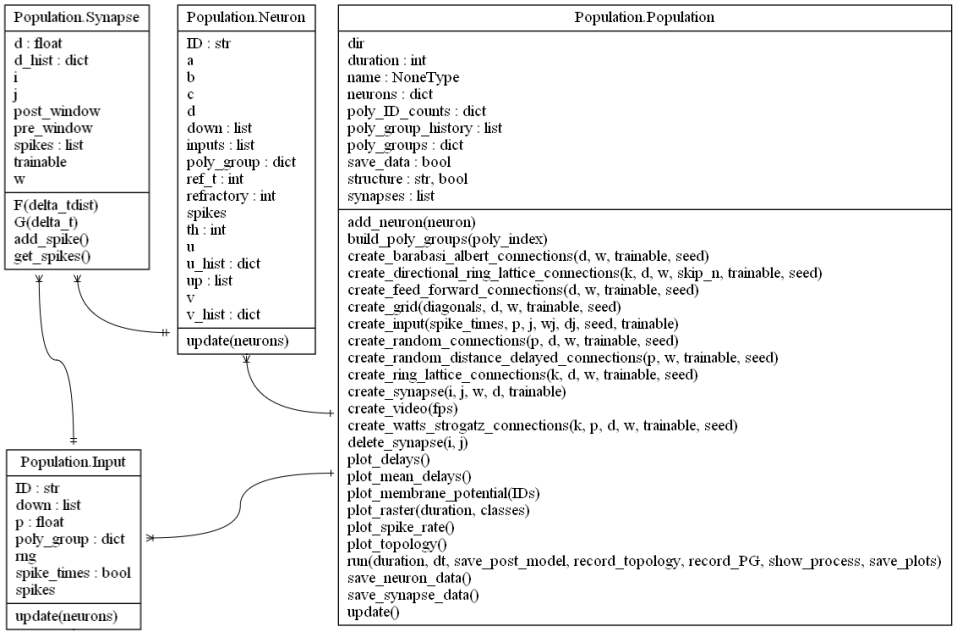

Figure 3.1: Class-diagram showing the most important components of the project code.

The neuron object contains of list of incoming synapses and one list of outgoing synapses. These synapses are shared objects between a pre- and post-synaptic neuron object. The update method of the neuron objects checks all incoming synapses for spikes, and integrates any spikes during the membrane potential update. If the spiking threshold is exceeded, a spike is added to all outgoing synapses. Part of the delay learning logic is also found in the neuron object. Specifically, it is the logic that determines which synapses should activate what part of the delay learning method. When a spike is generated, code logic determines the set of incoming spike that contributed to the spiking based on the spike time of the incoming spikes and the travel time between the two neurons. This give the arrival time of that spike, and if it is within a specified time window, it is a contributing spike. If a synapse contributes with multiple spikes it is the spike closest to the post-synaptic spike time that takes precedence. Likewise if there are no contributing spikes, logic checks if there are spikes that arrive withing another pre-specified time-window after the post-synaptic spike time, which triggers that second part of the delay learning method.

The synapse object is a structure that accepts and delivers spikes. Each spike is registered by its spike time and the synapses delay and weight at the time of spiking. When a post-synaptic neuron checks for incoming spikes, the synapse object checks for the condition $T == s_i(t) + d_{ij}(t)$, where T is the current simulation time, $s_i(t)$ is the pre-synaptic spike-times and $d_{ij}(t)$ is the delay between neuron i and j at time t. The synapse object also holds the two delay learning functions $F(_{dist}$ and $G()$, which will be detailed in section 4.1.

The framework includes a few features that have not been used in this project such as including all seven neuron types described in the paper of Izhikevich from 2003 [18]. This allows for creating heterogeneous populations of various neuron types. The framework also allows for a heterogeneous combination of trainable and fixed connection delays.

3.2 Transmission delay model

One of the most crucial parts of delay learning is the implementation of the delays themselves, which is not a trivial matter. A complex network consists of many neuron pairs that are connected by a theoretical axon. This axon has an associated delay, and the axon implementation must therefore ensure that a spike received in one end, reaches the other end in accordance with its delay. There can be multiple spikes in transit between two neurons at any time and they must reach the post-synaptic neuron at the right time and in the correct order.

One solution is to implement the connections as queues, were the pre-synaptic neuron would push a boolean value of true onto the que if it produces a spike, and false if not. In the same timestep, the post-synaptic neuron would retrieve a boolean value from the opposite end where a true value would increase its membrane potential by the connection weight and a false value would cause no change. This method of adding and retrieving from either end of the queue acts as the spike delay between the two neurons, and the length of the queue when initialized determines the delay. The queue is then initialized with all false values. The minimum delay allowed between neurons is equal to one time step, or dt, when using this method, which is both logical in relation to the iterative manner in which the simulation proceeds, and it also avoids the issue of retrieval from an empty queue. This queue method can be seen in figure 3.2.

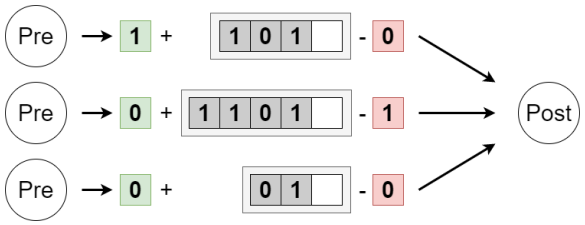

Figure 3.2: The que method where the pre-synaptic neurons add spikes, and the post-synaptic neuron retrieves them. The green squares indicate added spikes, while the red indicates retrieved spikes.

Although the method is efficient and simple to implement, it raises some issues when the delays are altered in real-time. A reduction of the synapses delay would warrant the removal of Boolean values from one of the ends of the queue, which in turn would alter the intra-neuronal communication. Additionally, the method requires that boolean values are added to the queue regardless of whether there is a spike or not, something that can slow down the simulation when there are many synapses.

Another method that avoids the needless adding of boolean values is the counter-method. Each time a neuron spikes, it adds a counter to each of its downstream connections initialized with their respective delay-values. These values are subsequently decremented for the following time iterations until reaching zero upon which the spike has arrived at the post-synaptic neuron leading to an increase in its membrane potential. If an axonal delay is changed, it is simply the initializing value that is changed, and the current counters are unchanged and no spikes are removed. This method is illustrated in figure 3.3.

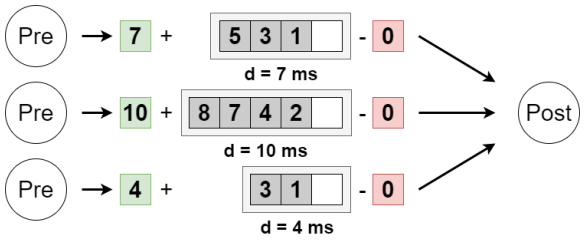

Figure 3.3: A counter is added to the axon each time a pre-synaptic neuron spike. The counters are decremented for each time iteration, until reaching zero, upon which the spike is integrated into the membrane potential of the post-synaptic neuron.

This method led to an issue with the inherent asynchrony of state updates. Since the order in which the neurons are updated is in practice arbitrary, the decrementation of a new counter during an iteration is dependent on whether it is added prior or posterior to the state updates of the post-synaptic neuron which is responsible for the counter updates. This results in delay-related errors.

An obvious solution is to use the global variable for time, which is accessible to all neurons. The method can be seen in figure 3.4. Whenever a pre-synaptic neuron fires, the time of the spike, in addition to the weight and delay at the time of spiking, is added to the axons list of spikes. The post-synaptic neuron checks all spike times in its upstream connections against the current time, taking into account the delay of each connection, and if it is equal, the spike has arrived at the post-synaptic neuron and can be integrated into its membrane potential.

This method ensures that the spike times and propagation of spikes are synchronized across the entire model. Since it is assumed that a spike cannot instantaneously arrive at the post-synaptic neuron, there is a minimum of one simulation iteration before a spike propagates to the end of a connection. This invalidates the issue with the sequence of delay updates, since any spike from the previous time step will be accessible to the post-synaptic neuron at the current time step.

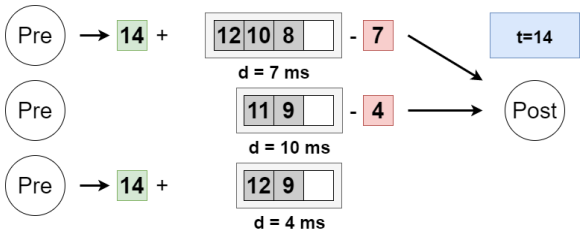

Figure 3.4: The time of the spike in addition to the weight and delay of the connection at the time of spiking is added to the axons list of spikes. The post-synaptic neuron checks the list of spikes in its upstream connections and integrates spikes that align with the current time when considering the delay value.

3.3 Izhikevich model implementation

The chosen neuron model for this project is the Izhikevich model detailed in section 2.2.3. This model, as previously mentioned, allows for multiple neuron sub-types depending on the choices for the constants a, b, c and d and is driven by two differential equations.

The simplest neuron type, and the one which will be explored in this work, is the Regular Spiking neuron, abbreviated RS neuron. This neuron fires a single spike when the membrane potential crosses the threshold of 30mV, followed by a refractory period where the reset of u, results in a depression of the membrane potential. This depression constitutes the refractory period of the neuron. As the increased value of u slowly returns to equilibrium, the membrane potential will also return to its resting state. The chance of eliciting a spike in the refractory neuron will be reduced, and depending on the strength of the input, the neuron might not fire at all during this time window.

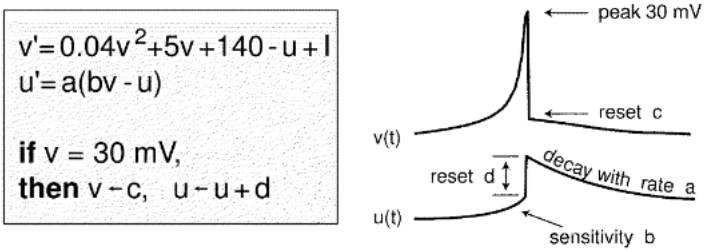

Figure 3.5: Constant b determines the sensitivity of $u(t)$ to the membrane potential $v(t)$ and a is the rate of change for $u(t)$ [18].

As seen in figure 3.5, constant b determines the sensitivity of the u variable to the membrane potential $v(t)$ and a is the rate of change for $u(t)$. c and d is the reset values for v and u respectively. The initialization of the u and v varies, with -65 mV used in one code example [22], while another code example [23] uses -70 mV. Testing showed that for the RS neuron, an

initial membrane potential of -70 mV and a u-value of -14 results in a stable state for both values.

The choice of implementing the Izhikevich neuron model is accompanied by several design choices that have varying degrees of impact on the dynamics and behavior of the individual neurons and the system as a whole. These choices include, for example, the order of state updates, the manner in which the v-variable is updated, the choice of simulation resolution and the integration of spikes. The order of state updates is important in order to avoid incorrect values, such as v must be updated before u. Reducing the simulation duration might be beneficial, but could also result in unexpected behavior. Likewise, the method for integrating spikes is important, especially if the time resolution is changed from the original implementation [18], which uses 1 ms.

As mentioned, the order at which the system states are updated requires some afterthought. The state updates and general procedures for simulation are implemented according to the pseudo-code below:

```
while t < duration do
    for neuron in neuron_list do
        Integrate spikes;
        update v and u;
        Limit v to threshold (for plotting);
        Plot v and u;
        if v == threshold then
            Register spike;
            Reset v and u;
            Trigger learning mechanism;
        end
    end
    t+=dt;
end
```

Algorithm 1: General simulation procedure

By updating the v- and u-variable before checking and registering spikes ensures that the spike-time is correct in terms of when the time-step in which the membrane potential crosses the threshold. The value for v is allowed to exceed the threshold for updating the u-variable but is then limited to the threshold value for plotting purposes. This ensures that the membrane potential at the time of spiking is even along the entire simulation, while not affecting the dynamics of the neuron as the membrane potential is reset right after its values is registered.

Since the membrane potential can only be recorded once per time step, and its value is potentially updated twice; once during a regular update, and once during a potential reset following a spike, it is logical to register it prior to the reset in order to include the entire form of the spike. Consequently, the reset value might not be registered as the membrane potential is again updated in the next iteration before it is stored. This

can be considered a trivial matter, however, as it is far more important to visualize the spike peak, than the exact reset value.

In one of the Matlab code examples by Izhikevich [22], he has opted to check and register spikes at the start of each iteration before state updates, thus the spike is registered one timestep after the membrane potential crosses the threshold which introduces a delay in the system further complicating its dynamics. This is unwanted behavior in this project, and the sequence of procedures is therefore as described above.

An interesting feature of the same code, however, is the separation of the v-updates which is done in two sequential steps, where each step is multiplied by a factor of one half. Izhikevich explains this as a method for decreasing numerical instability. Although the instability seen in the Brian2 library in high-activity networks was not observed in the new code, testing with $\mathrm{d}t = 1$ ms, showed that omitting this step could lead to oscillations of the membrane potential at the equilibrium state for high inputs, which can be seen in figure 3.6. This is uncharacteristic behavior for the regular spiking (RS) neuron model used in the test. An interesting observation is that this behavioral artifact was not present when $\mathrm{d}t = 0.1$ ms.

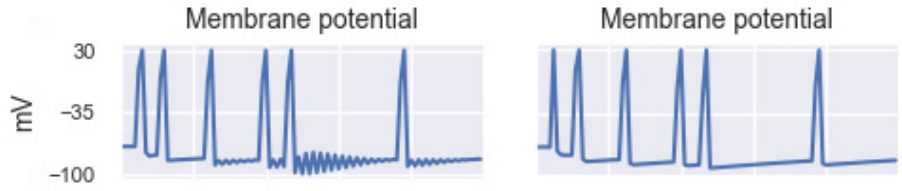

Figure 3.6: Left plot shows the oscillatory behavior of a single regular spiking neuron exposed to high inputs. Right image shows no oscillations when v-updates are performed in two steps.

Both in the Izhikevich [18] paper that introduces the neuron model and in its accompanying code examples [22], the time-step used is $\mathrm{d}t = 1$ ms. Differential equations can be fine-tuned to specific time steps, but this is not explicitly mentioned to be the case for these equations in the paper. An obvious motivation for decreasing the time step is that it typically decreases the integration error which is a compounding error that will increase with simulation duration. Another motivation for reducing the time step is that it increases the resolution of delays, potentially increasing the repertoire of dynamic behavior of the models. In order to assert the validity of the model at lower time steps, a comparison is crucial.

This comparison can be seen in figure 3.7, which shows very similar behavior. The smaller time step is achieved by relying on the Euler integration method given by equation 3.1,

$$y_{1+n} = y_n + hf(t_n, Y_n) \tag{3.1}$$

where y_{n+1} is the value of the variable in question, in this case either v or u, for the current time step. y_n is then the value for the previous time step. h is the size of the time step, referred to as $\mathrm{d}t$, which is multiplied with the

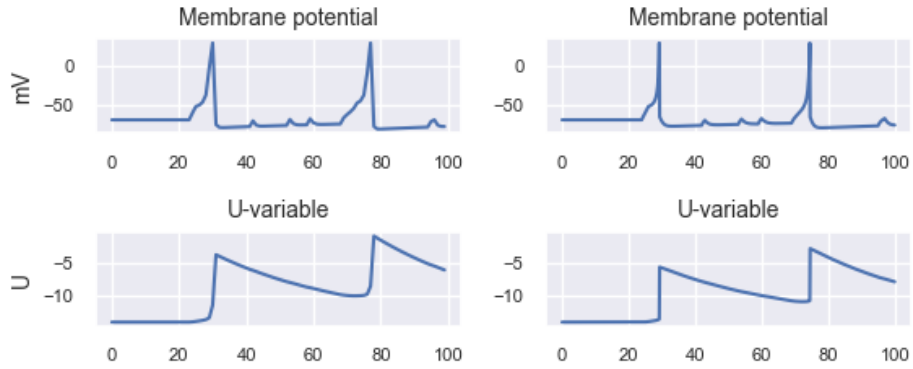

Figure 3.7: Comparison of the v- and u-variables of a single regular spiking neuron with dt of 1 ms in the left column and 0.1 ms in the right column. The neurons in both examples are exposed to identical input.

differential equation describing the change of y. Thus, the updates for the neuron model are given by equation 3.2 and equation 3.3.

$$v_{t+\mathrm{d}t} = v_t + \mathrm{d}t(0.04v^2 + 0.5v + 140 - u + I) \tag{3.2}$$

$$u_{t+\mathrm{d}t} = u_t + \mathrm{d}t(a(bv - u)) \tag{3.3}$$

In practice the v-update is as mentioned performed in two identical steps where each operation is multiplied by 0.5.

An important consideration, however, is the manner in which I is integrated. For the time step of 1 ms, I is simply added to the membrane potential since dt $=$ 1 ms. For a time step of 0.1 ms, the effect of I will be one tenth if it is simply applied for one time step only. Removing the I-variable from the equation and adding it prior or posterior to the differential equation will not elicit the same behavior and will not be mathematically equivalent. It is therefore necessary to integrate the input over the same time-period as for the larger time step. In other words, the input must be integrated over 1 ms, and therefore applied for 10 iterations when d$t = 0.1ms$.

It is worth noting that there is a slight time-shift between the two plots in figure 3.7, where the simulation using dt $=$ 0.1 ms, results in a slight delayed reaction to inputs. The reason for this is that when the spike reaches the post-synaptic neuron the input is gradually integrated from that time point and over the ten next iterations as mentioned, but when dt $=$ 1 ms, the input is instantaneously integrated during the time step in which it reaches the post-synaptic neuron. In other words, decreasing the time step introduces another source of delay, but by decreasing dt a higher resolution for delay values is possible.

3.4 Neuron behavior

In order to be able to configure larger networks of Izhikevich neurons, it is important to understand the properties and behavior of single neurons. These behaviors includes, how much input is required to elicit a post-synaptic spike, how the magnitude of the input affects the response in the post-synaptic neuron, how the alignment of incoming spikes affect post-synaptic response, how long the intrinsic refractory period of the Izhikevich neuron is and finally how quickly the neuron returns to its equilibrium state after an weak input. Based on these properties, better choices can be made for model parameters such as connection weights and delays, network connectivity and input frequency in addition to choices regarding the learning method implementation.

3.4.1 Neuron response

A single neuron was exposed to varying inputs, and its minimum input required to cause a spike was found to be a single spike with a weight of 16.4 when dt $=$ 1 ms. With such a weak input, the post-synaptic spike occurred 11 ms after the pre-synaptic spike arrived at the neuron. This period will be referred to as the neuron response time in this work. Decreasing the step size of the simulation to 0.1 ms, doesn't elicit a spike until the weight is 16.8, with a corresponding neuron response time of approximately 9 ms. Further testing with this higher time resolution shows a steady decrease in response time as the connection weight is increased. When the weight is approximately 40, the neuron response time starts evening out. Figure 3.8 shows the neuron response time for a relevant weight interval for both time resolutions.

This shows that the temporal dynamics of the model is not only dependent on the delays of the connections, but also on the magnitude of the input, which is again dependent on the number of incoming connections that provides a spike and their respective weights.

These interesting observations can be further explored by looking at spikes from two incoming connections with various connection strengths and spike offsets. Figure 3.9 shows spikes in blue and dormancy in red for each combination of spike shifts and connection weights. The y-axis shows the shift in milliseconds between the arrival time at the post-synaptic neuron, and the weight is the connection strength of both synapses. The left plot is for $dt = 1$ ms, while the right plot is for $dt = 0.1$ ms. There is a clear phase transition going from no spike to spiking which is dependent on the aforementioned variables. The time-window within which two spikes must arrive to elicit a post-synaptic spike increases somewhat exponentially with an increase in connection weight.

The input shift and weight not only determines the occurrence of a spike, it also determines the time it takes for the membrane potential to cross the spiking threshold. As previously mentioned, this is referred to as the neuron response time. Plotting the individual response times for various weights and input shifts in figure 3.10, we can see that there is an

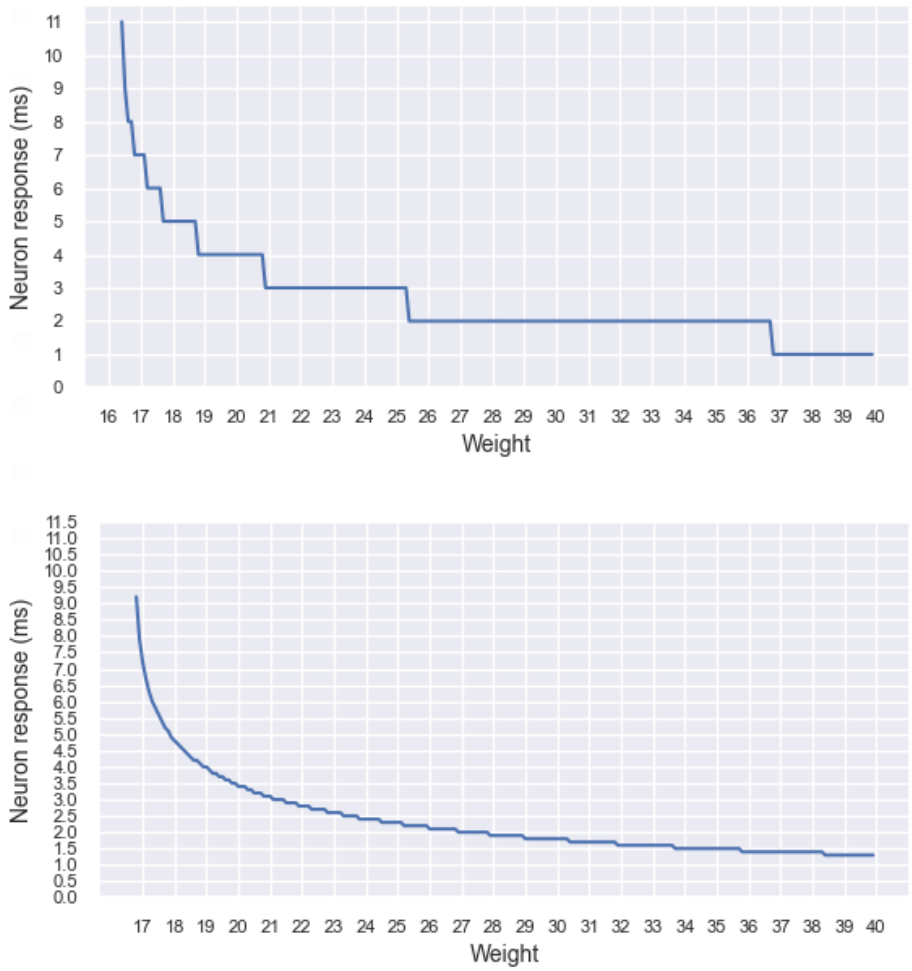

Figure 3.8: Neuron response for a given weight. The top plot uses d$t = 1$ ms, while the bottom plot uses d$t = 0.1$ ms.

exponential relationship between shift and response time. The choice of weight interval covers all integer weights that elicits a post-synaptic spike as a pair of two incoming spikes.

The most dramatic changes to the response time occurs as the shift in input reaches its maximum value that still elicits a spike. Larger weight values, which have a wider interval of possible shifts that still elicits a post-synaptic spike, will have a more consistent response time.

An interesting observation is that the maximum response time before spiking stops does not seem to follow a clear trend. This can be seen going from a weight of 12 to a weight of 13, which results in a lower response time, followed by a higher response time when the weight is 14. This is likely a bi-product of the simulation step size which limits the size of the shifts to 0.1 ms.

For initial configurations of delay learning models, the connection weights should be carefully chosen with these results in mind, with lower

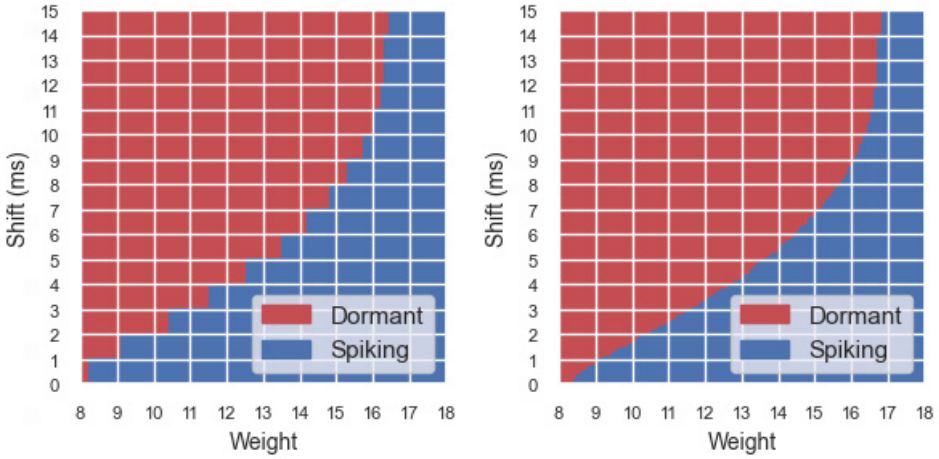

Figure 3.9: Red indicates no spike and blue indicate spike at the post-synaptic neuron for a given connection weights of two connections and their spike off-sets to each other. The left plot is the result of $\mathrm{d}t = 1$ ms, while the right plot uses $\mathrm{d}t = 0.1$ ms.

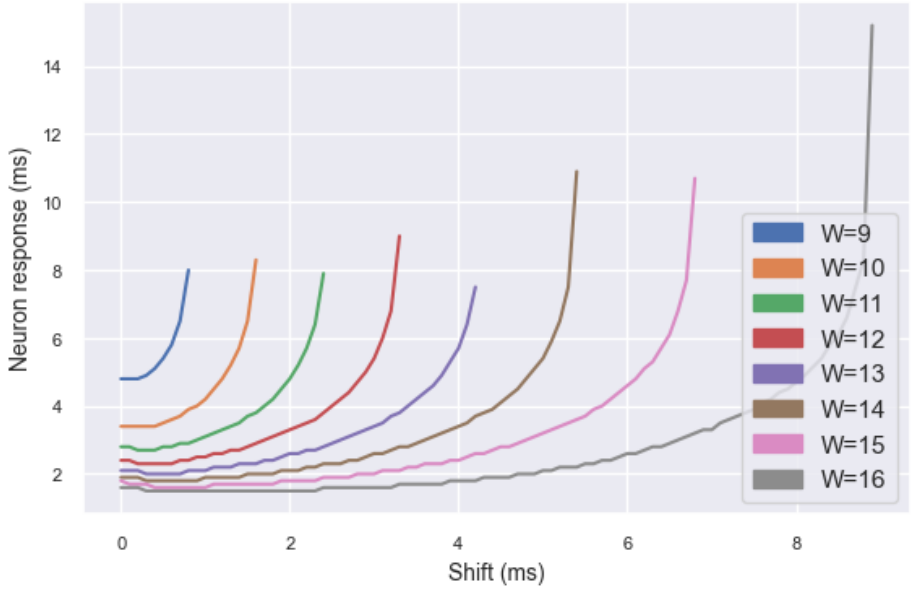

Figure 3.10: The neuron response time for different spike alignment shifts for all integer weights that can elicit a spike. Neuron response time is calculated from the time of arrival for the last spike to spike time of post-synaptic neuron and is for $\mathrm{d}t = 0.1$ ms.

weights requiring a higher precision of arrival time at the post-synaptic neuron to elicit a post-synaptic spike, possibly reducing chaotic behavior. However, lower weights will also increase the reaction time of the neurons membrane potential, introducing more delay in the network. Choosing

higher weight values gives more consistency as the response time changes less with input shifts, but the large spike alignment window could lead to more chaotic behavior.

3.4.2 Refractory period

An important aspect of the the neurons behavior is how it responds to changes in membrane potential. This is an important consideration because it indicates how long a neuron is affected by previous inputs. This can help determine the proper interval between input patterns, if it is desired that the consecutive activity does not interfere with each other.

There are two specific cases that are of interest, namely the refractory period following a spike generation, and the return to equilibrium following an excitement of the membrane potential without spike generation.

The implementation of a forced refractory period, where incoming spike are ignored, was tested early on, but keeping with the notion of trying to implement the Izhikevich model in its original form, this was not used for the simulations presented in this project. Instead, the simulations rely on the inherent refractoriness of the model itself. This means that it is possible to elicit a spike during the refractory period if the combined input is strong enough.

The duration of the refractory period in this case can be considered to last from the time-point of the initial spike until the earliest spike arrival time that can lead to spike generation. This is a more useful definition as defining this period as a spike-to-spike interval gives little information about when the neuron is capable of receiving input that can lead to spiking. Based on this definition, and the use of a connection strength of 32, equivalent to perfect spike alignment of two connections of strength 16, a time window of 22ms is required before a spike can arrive and ultimately lead to another spike generation when the time step is set to 0.1 ms. Naturally, the neuron response time is slightly increased as the membrane potential has not reached its equilibrium state entirely. Additionally, the refractory period is increased for imperfect spike alignment. For a spike pair of the lowest alignment that still can elicit spike, which is an offset of 8.9 ms, a period of close to 450 ms is needed for this spike pair to cause a post-synaptic spike. The practical duration of the refractory period is therefor quite significant. A shorter refractory period would be preferred, but in order to preserve the integrity of the model, no changes will be made.

Figure 3.11 shows the phase transition going from dormant to spiking as the offset decreases and the interval between the initial spike and the following input increases enough to allow a new spike to be generated. An interesting observation is the occurrence of spiking for low intervals and offsets, barely visible in the lower left corner of the plot. Further investigation showed that there is a small time-window following an initial spike, where the membrane potential is reset to -65 mV according to the c parameter of the model, when spiking is possible. Due to the drastic change in the u-variable, this time-window is quickly followed by a further decrease in membrane potential below the equilibrium state of -70 mV,

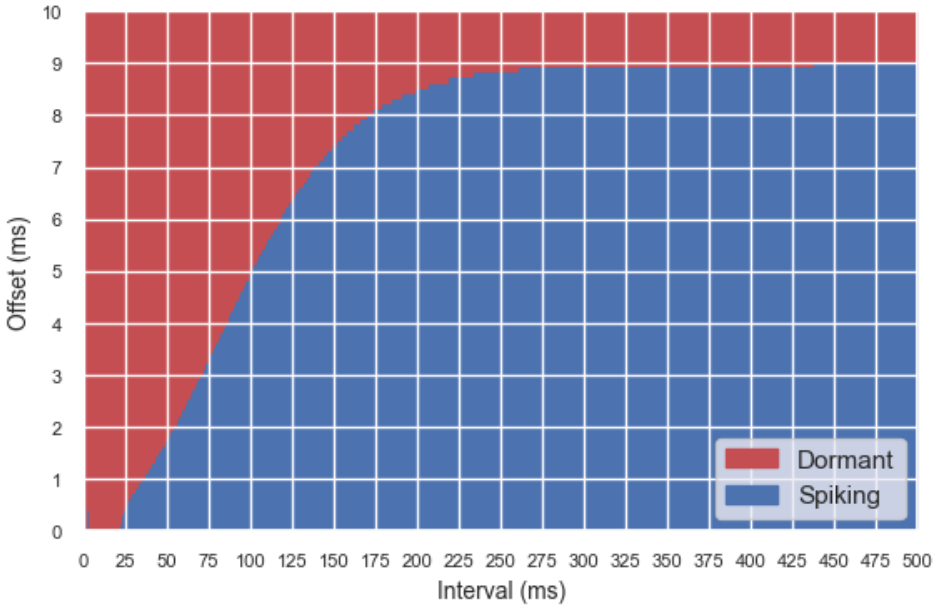

Figure 3.11: Plot showing whether a second spike occurs at a specific interval, on the x-axis, following an initial spike as a result of two inputs with a weight of 16 and an offset described by the y-axis.

which prevents spiking. This issue can be avoided by reducing the reset value of the membrane potential, but as previously mentioned, it is desired to retain the integrity of the original model as much as possible.

Another typical scenario that can be encountered is the arrival of a single spike which elevates the membrane potential without causing a spike to occur in the receiving neuron. The membrane potential slowly returns to its equilibrium state, and during this time, the neuron is more susceptible to spiking when new input is given. If successive inputs are given and it is a requirement that these inputs do not affect each other, it is important to know the extent of the time window of this elevated state of the membrane potential. To test the this hypothesis, an input of 16 is used, which is close the the maximum input that will not lead to a post-synaptic spike. The membrane potential returns to within 0.1 mV after approximately 11.1 ms.

An interesting observation is that following the return to equilibrium the membrane potential overshoots below the equilibrium state of -70 mV, effectively leading to a weak refractory period. This requires a slightly elevated combined input of 17.7 in order to elicit a post-synaptic spike, at the peak of this small refractory period which occurs at 16.5 ms following the onset of the previous input. Due to the inversely exponential change in the membrane potential, the return to within 0.1 mV of equilibrium following the onset of the input when including the additional refractory period is significantly longer at 69.3 ms.

Based on these observations it is difficult to completely avoid interference between inputs. If inputs are spaced out at around 300 ms intervals,

there should be little interference.

3.5 Polychronous groups

In the 2006 paper by Izhikevich [21], polychronous groups are described as groups of spiking neurons that exhibit reproducible time-locked firing patterns, and that the number of coexisting polychronous groups can far exceed the number of neurons in a network. This indicates a significant ability for computational capacity, and inspired the classification method used in this work, as will be shown in section 5.1.

In order to test the concept of polychronous groups, the theoretical models described by Izhikevich was implemented using real networks of the Izhikevich model. These theoretical models consists of five neurons and use handpicked delays to maximize the number of coexisting polychronous groups.

The handpicked delays and the 14 polychronous groups can be seen in figure 3.12. All simulations in this sections used $dt = 1$ ms which appears to be the time resolution used in the Izhikevich paper.

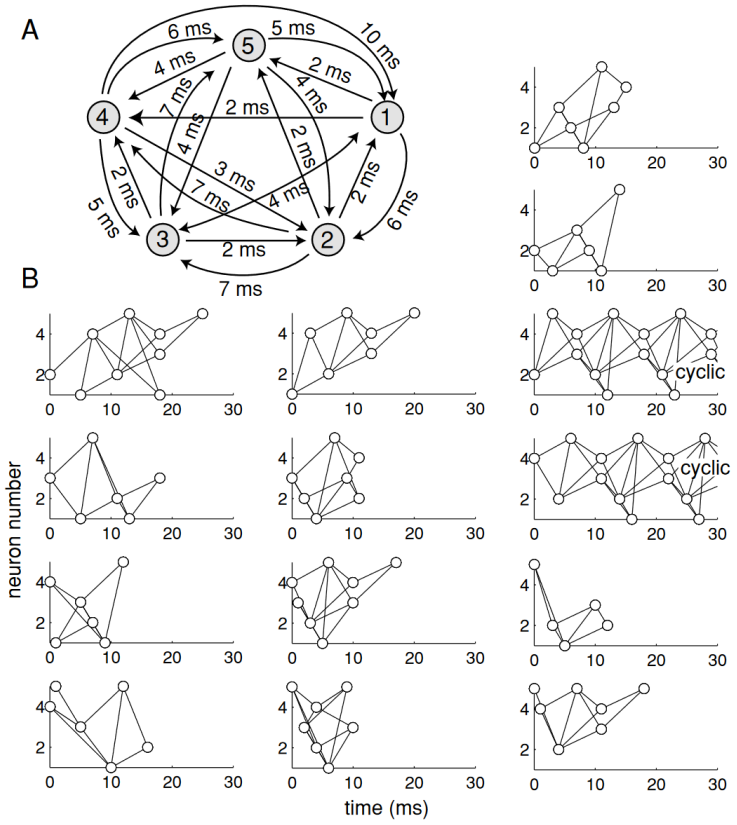

Figure 3.12: Polychronous groups achieved by hand-picking delays for a fully connected network of 5 neurons [21]

When implementing these specific delays with the intention of recreat-

ing the polychronous groups, the results were varied. The activity patterns seen in part B of figure 3.12, is the result, as mentioned in the paper, of a toy-model, and as such does not take into consideration the complex dynamics of the Izhikevich neuron model. The discrepancy between the results from the toy-model and the recreated configuration using the Izhikevich neurons can therefore be explained by the responsiveness of the neurons. In the toy-model, the neurons responds immediately when two incoming spike coincide at the post-synaptic neuron. With the Izhikevich model, however, the response is never immediate. The lower the connection weights are, the more the pattern gets distorted and stretched as each process of spike integration adds a delay to the dynamics of the spike propagation. With high connections weights, a higher resemblance to the reference pattern can be achieved, but at the cost of noise, as neurons no longer require a single spike to reach threshold.

In order to reduce the delay that is introduced by the neuron, some care should be taken when choosing the connection weights. In fact, when looking at the neuron response plots in figure 3.8 for $\mathrm{d}t = 1$ ms, the highest connection strength that does not elicit a post-synaptic response with a single spike is 16.3, which equates to an input of 32.6, for perfect spike time alignment at the post-synaptic neuron. This gives a response time of 2 ms, from the time at which the two spikes arrive at the post-synaptic neuron until the subsequent spike is produced.

The neuron response time per millisecond offset in spike alignment for weights in the interval $w = [13, 16]$ can be seen in figure 3.10. This is the set of integer valued weights where the response time is 2 ms, and where no single spike can elicit a post-synaptic spike. It could be beneficial to choose a value in the lower end, as this will minimize the time in which non-spiking neurons returns to equilibrium after receiving spikes, consequently reducing noise. A connection strength of 13, should be sufficient in this regard, but testing shows that with such weak connectivity, consecutive spikes from single neurons are prevented by the refractoriness of the Izhikevich neuron for the time scale relevant for the polychronous examples of figure 3.12. The shortest duration between consecutive spikes in a single neuron in the toy example is about 8 ms. Increasing the weight to 14 and 15 allowed for consecutive spikes within the 30 ms simulation, but not in the quick succession required to recreate the polychronous patterns. We are therefor left with choosing the maximum weight that still prevents spike elicitation from single connections. For simplicity's sake the weight can rounded to $w = 16$. Isolated tests of a single neuron receiving perfectly aligned spikes from two such connections showed that a inter-spike time of 28 ms was required before a new spike could be produced. This is consistent with the observations made in section 3.4.2, with the slight difference likely being a result of the different time resolutions.

This shows that although subsequent spikes from single neurons are observed in networks consisting of weights of 16 within the relevant time frame, they are likely the product of more than two spikes. Based on these observations it can be concluded that it is essentially impossible to recreate the polychronous patterns using the Izhikevich neuron model on the same

time scale.

It is still worth exploring the result of the simulations attempting to recreate the patterns however, as it can uncover more aspects about the application of Izhikevich neurons in networks. According to figure 3.10, shifts up to $10ms$ will still elicit a spike for a connection weight of 16, so a lot of noise should be expected. With this knowledge about the dynamics of this particular set of parameters, a more thorough understanding of the output for the hand-picked connections delays can be made.

It is worth noting that there appears to be some inconsistency in the plots in figure 3.12. Some plots seem to include a $1ms$ spike response delay for some neurons. It is unclear whether this is due to the difficult task of hand-picking delays, or if some margin of error is allowed for spike alignment as is the case for real spiking networks. This has a significant effect on the implementation using the Izhikevich neuron models, as different spike alignments introduces shifts in spike times that propagate throughout the network. It is therefore beneficial for the sake of simplicity to choose an input pattern that adheres to the simple immediate response mechanism, as is the case for input pattern 3 in the first column of figure 3.12.

Figure 3.13 shows the theoretical sequence of activity based on the observations made using the specified parameters. The smaller values shows the delays between neurons, identical to the ones described in figure 3.12. However, the larger numbers indicates the arrival time of the pre-synaptic spikes in parenthesis, and the delayed spike time of the post-synaptic neuron. Neuron 3 receives spikes from neuron 1 and 4 with perfect alignment, and according to figure 3.10, will have a neuron response time of 2 ms, resulting in a spike time at $t = 7$ ms. Likewise, neuron 2, receives spikes with an alignment shift of 2 ms, resulting in a response time of 2 ms. The next spike is at neuron 1, and in this case, we disregard the observation that neuron 1 is unable to produce a second spike from two inputs due to refractoriness. The spike at neuron 1 occurs at $t = 15$. Finally a spike at neuron 5 happens at $t = 19$ ms.

Figure 3.14 shows the resulting spike times when implementing the parameters discussed. The initial two spikes from neuron 1 and 4 achieves correct spike times through some implementation specific measures so as to start the spike sequence of correctly. The third spike occurs at $t = 7$ ms for neuron 3, as predicted. The next spike from neuron 5 at $t = 8$ ms, however seems to be incorrect as it should not occur until $t = 19$ ms according to 3.13. Looking at the delay configuration of figure 3.12, neuron 3 has a 7 ms delay to neuron 5 excluding it from having any effect in relation to this specific spike, as it would arrive after the post-synaptic spike. However, neuron 4 has a connection delay to neuron 5 of 6 ms. Likewise neuron 1 has a delay of 2 ms, thus their spike alignment offset is 3 ms, which is within the window that will elicit a post-synaptic spike 2 ms after the arrival of the last spike. This causes neuron 5 to spike at $t = 8$ ms, which is consistent with the result in figure 3.14.

Similarly, the spike at neuron 2, should theoretically happen at $t = 11$ ms, but occurs at $t = 9$ ms. This is likely due to an additional input from

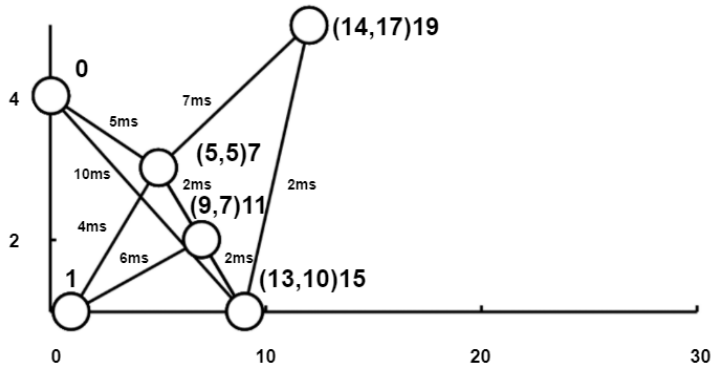

Figure 3.13: Smaller numbers at the edges indicate delays in milliseconds, while larger numbers by nodes indicate spike arrival time for the two incoming spikes in parenthesis followed by spike time in milliseconds.

neuron 4, not considered in figure 3.13. Neuron 4 is connected with a delay of 3 ms to neuron 2. Neuron 3 connects via a 2 ms delay and neuron 1 connects via a 6 ms delay. This gives spike arrival times of 3 ms, 9 ms and 7 ms respectively.

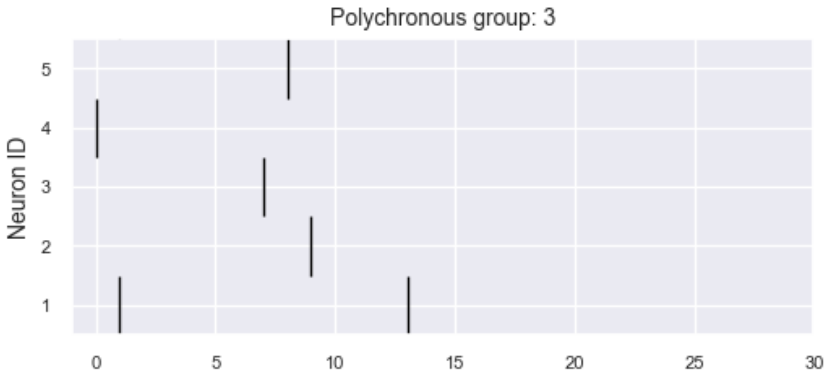

Figure 3.14: Shows the simulated spikes for the input pattern of polychronous group 3.

Since a weight of 16 allows for spike offsets of 10 ms while still producing a post-synaptic spike, it is clear that all the spike times mentioned is well within this window. However, the two first spikes to arrive gives a response time of 2 ms, eliciting a spike in neuron 2 at $t = 9$ ms, thus the third spike is not necessary for the creation of this spike. This observation is consistent with the results in figure 3.14.

The next spike that is expected to occur is at neuron 1, at $t = 15$ ms. However, due to the change in spike time at neuron 2, the arrival times are 11 ms and 10 ms from neuron 2 and 4 respectively. The expected spike time would therefore be at $t = 13$ ms instead, which is consistent with figure 3.14. The next expected spike should happen at neuron 5 at $t = 19$

ms but never occurs. When considering the shifts of previous spike times the arrival times at neuron 5 is $t = 15$ ms and $t = 14$ ms from neuron 1 and 3 respectively. This would lead to a new spike time at $t = 17$ ms. The missing spike can be attributed to the refractory period following the previous spike.

Figure 3.15 shows the membrane potential of neuron 5. The membrane potential prior to the initial spike at $t = 8$ ms, is -70 mV. Following the spike there is a noticeable decrease in membrane potential to about -80 mV, which is 10 mV less than its equilibrium state. This prohibits the following inputs from causing a spike. This is not possible to mitigate as increasing synaptic strength further will allow for spikes to occur as a result of single input spike.

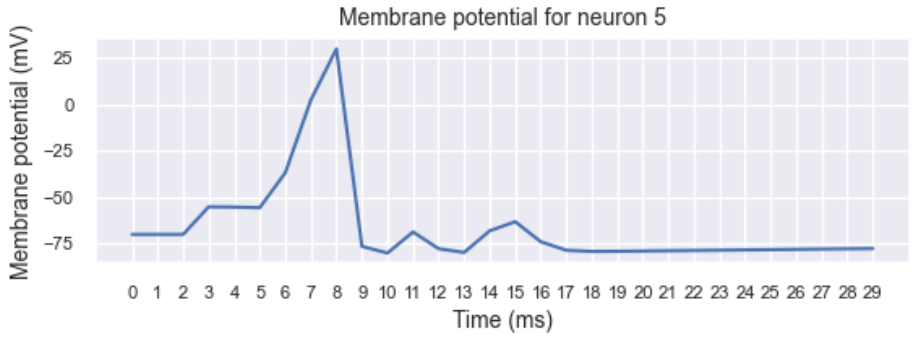

Figure 3.15: Initial membrane potential before spike is -70 mV. After spike, the membrane potential dips to -80 mV, slowly returning towards the initial equilibrium state. The ability for the neuron to spike is reduced while returning to its normal resting potential.

Similar analysis of the other polychronous patterns can be made, but the concept remains the same. In theory, the patterns can only be reproduced if there is an equal number of neurons in every path of any spiking neuron linking it to the instigating neuron of the spike pattern, and this does not consider the noise of other spiking neurons which would alter the result. And in that case, it is only the order which can be reproduced and not the exact spike times. Whenever there is a non-zero time component for spike integration, there is a possibility that delays will be introduced into the information propagation asynchronously throughout the network which will alter the timing and spike arrival times. Additionally, the refractory period which is an intrinsic property of the Izhikevich neuron, as shown in figure 3.15, will reduce the ability of neurons to spike over a time scale that exceeds the duration of the polychronous patterns. Lastly, the relatively large time window in which spikes can arrive and elicit a post-synaptic spike, allows spikes from more neurons to interfere and alter the spike time of the post-synaptic neuron. This is not a trivial matter as it introduces another layer of complexity when considering these recurrently connected networks.

Although the model can not reproduce the pattern from the ideal model, the output is just as valid as polychronous groups since they are reproducible and activated through a specific input pattern.

Chapter 4

Delay learning

This chapter will cover the process of developing and validating the delay learning method, which is the main contribution of this project. This includes the assumptions and reasoning behind various design choices, as well as the formulation of the final method. A short demonstration of the learning method applied to a simple network is then given.

This is followed by a section describing three distinct classes of input patterns, which in turn was exposed to various simple networks in the final section of the chapter. This is done in order to analyze the behavior of the delays when exposed to different types of inputs. The final result section is preceded by a detailed explanation of the method for delay categorization.

4.1 Conceptual foundations for delay learning

When designing the local learning rule, one must consider various aspects of the model itself, such as how the activity of the network should be interpreted and extracted from the network. In the case of this project, the output of the network was extracted through the identification of polychronous groups. The specific implementation of PG detection is detailed in section 5.1. The general concept however can be illustrated by figure 4.1, where three neurons are connected with different delays to two post-synaptic neurons. Depending on the spike times of the three neurons, they can activate one of the two post-synaptic neurons if their spikes align. Although they are quite simple, this would constitute two distinct PGs.

This is an example of a simple network, but when the concept is applied to a larger network, similar interactions would occur many places in the network and lead to a distinct activity pattern that depends on the initial input.

Based on this method for output interpretation, the focus of the learning mechanism should be to consolidate the network activity associated with similar inputs that constitute a distinct input class. If one considers all instances of a particular input class, there likely exist some average or arch-typical class instance which the network should adjust for such that the activity of all instances within that class falls within some margin of error and is then correctly classified. This would be achieved through

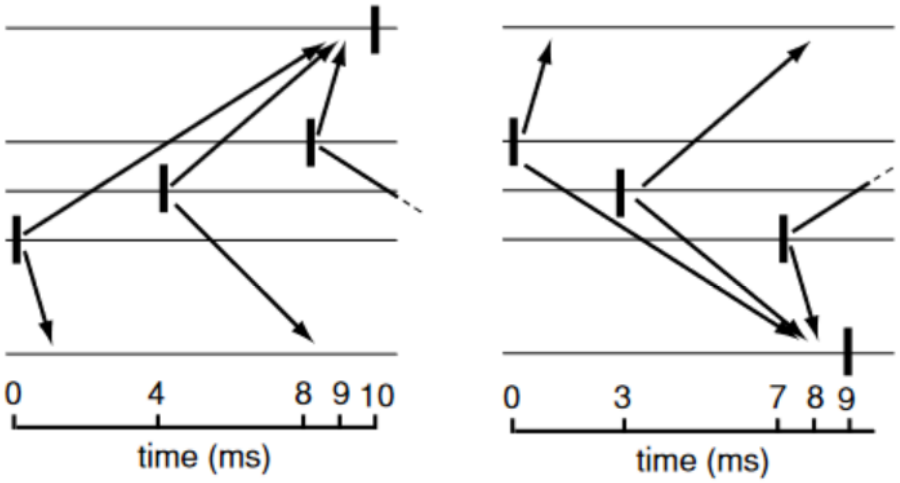

Figure 4.1: Three pre-synaptic neurons activating different post-synaptic neuron based on their order of activation [21].

incremental adjustments of delays for each instance, and after enough iterations, an optimal set of delays for this particular class is hopefully achieved.

This idea is visualized in figure 4.2. The figure shows one instance of an input class in each column. These three instances have slightly different spike times, and the desired behavior is for the network to respond similarly for all instances of the class by adjusting delays through delay learning. The inputs are given to the same network sequentially from left to right. The bottom row shows the corresponding incremental changes after each input, which gives that starting delays for the next input. The colored vertical lines represent the optimal spike arrival time for the particular class instance. Over time, these incremental changes will add up to some average change that give the optimal set of delays for this input class. Based on these observations, a delay learning method that attempts to align pre-synaptic spike times is proposed.

4.2 Activity-dependent delay learning method

The general mechanism of the learning method is inspired by the traditional STDP rule described in section 2.3.2, where a synapse is strengthened if the pre-synaptic spike results in a post-synaptic spike, and a synapse is weakened when the pre-synaptic spike is not contributing to a post-synaptic spike. In a similar fashion, the causal relationship between pre- and post-synaptic spikes will determine the changes that are made to the connection between them.

Since the increase in membrane potential caused by an action potential from a pre-synaptic neuron gradually decreases with time until the effect is completely diminished, there is a time window in which the pre-synaptic neuron has contributed to the increased membrane potential of the post-

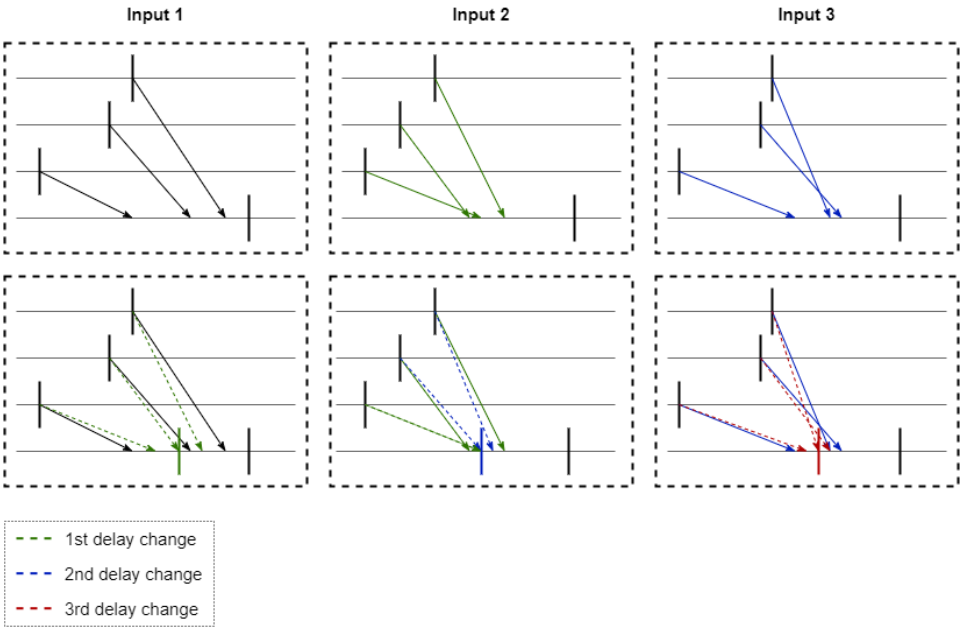

Figure 4.2: Three separate instances of a class is given sequentially from left to right, resulting in incremental changes to the delays seen in the bottom row. Vertical lines represents spikes, and the colored vertical lines indicate the optimal time for the set of spikes to arrive.

synaptic neuron as shown in section 3.4.2. Consequently its contribution is at its peak at the end of the 1 ms integration window starting at the instance the action potential reaches the post-synaptic neuron. Inversely, one can say that for any action potential produced by the post-synaptic neuron, there is a time window of influence prior to its spike time where all connections that contributed in this time window should be subject to the delay learning mechanism.

When considering the adjustment of delays instead of weights, the temporal dynamics is more complicated. A weight change will likely result in a shift in the post-synaptic spike time as the intensity of the input is changed. However, changes to delays will adjust the post-synaptic spike time as the result of both the realignment of spike arrivals increasing input intensity, and potentially the shift in average arrival time. There are three obvious ways of considering the delay adjustments, each one aiming to align spike arrival times. All three methods increases the likelihood of eliciting a post-synaptic spike, however they differ in the amount of change in the post-synaptic spike time, and in the average changes to delays.

Figure 4.3 shows the three ways that the pre-synaptic spikes can be aligned. Each example uses the same network, consisting of three pre-synaptic neurons connected to one post-synaptic neuron. The spike times of the pre-synaptic neurons are indicated by the black vertical lines, the green lines indicate the arrival time of the pre-synaptic spikes at the post-synaptic neuron and the spike of the post-synaptic neuron is indicated in blue. Additionally there is some time added between the last arrival

time and the post-synaptic spike as there is some response time before the neuron spikes as observed in section 3.4.1.

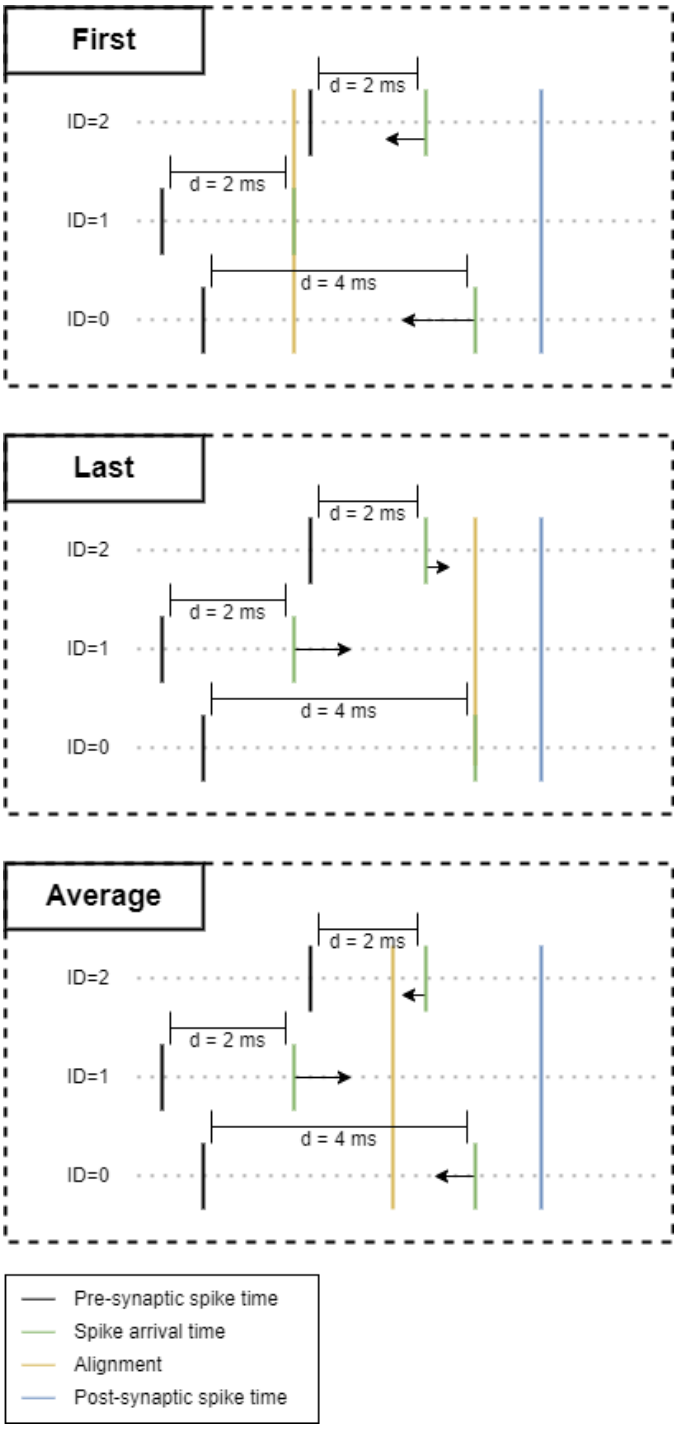

Figure 4.3: Black vertical lines indicate spikes from pre-synaptic neurons, while green vertical lines indicate the spike arrival time at the post-synaptic neuron when considering their respective delays. The blue vertical line signifies the spike time of the post-synaptic neuron.

The first method for grouping the spikes is to align them at the first arrival time, which occurs at neuron 1 and can be seen in the top plot of figure 4.3. The desired spike arrival time is indicated by the yellow line. This method leads to little to no delay adjustment to the middle neuron, a small adjustment to the top neuron and a larger adjustment to the bottom neuron. This change would decrease the spike time of the post-synaptic neuron the most. This method can also suffer from an issue where it is not possible to align all arrival times. This would happen if one neuron has a spike time that occurs after the earliest spike arrival time, as the delay can not be negative. This is the case in the example given, where the arrival time of neuron 1 occurs before the spike time of neuron 2. This method would also exclusively reduce delays, which could lead to delays saturating towards the lower limit, consequently limiting further learning in the network.

The second method aims at aligning spikes at the last spike arrival time, seen in the middle plot of figure 4.3. All changes to delays would therefore be increases. This method would likely result in the least adjustments to the post-synaptic spike time as there is a combination of positive shift as a result of increased delays, and a negative shift as a result of better spike alignment. This would result in a reduction in the downstream effect on other neuron interactions that are a result of this spike. However, as was the case with the previous method, the downside of exclusively positive changes to delays, is the problem of delays saturating to to limits of allowed delay values. This method would also introduce a lot of long delays into the system, which might eventually lead to slow computation times.

The last method, seen in the bottom plot of figure 4.3, aims at aligning the spikes at the average spike arrival time for all axons involved. This would involve increasing certain delays, while decreasing others. Although, the average arrival time would remain the same, the post-synaptic spike time would still see a shift, as the offset between spikes is reduced resulting in a greater immediate post-synaptic response. This method works better at avoiding delay saturation, and will likely not introduce significantly more system delay.

Based on these observations, the last method was determined to be the best option. Since the concept is based on gradual incremental changes, it was important to include some proportionality in the delay changes based on the distance between spike arrival time and the average spike time, denoted as Δt_{dist}. In other words, spike arrival times furthest from the average spike arrival time should experience the largest delay change, while closer ones should experience smaller delay changes. A reversed sigmoidal function was deemed a good choice for this learning method and is given by equation 4.1,

$$F(\Delta t_{dist}) = \frac{a}{1 + e^{\Delta t_{dist}/b}} - \frac{a}{2} = -\frac{a}{2} tanh(\frac{\Delta t_{dist}}{2b}) \tag{4.1}$$

where $\Delta t_{dist} = (t_{pre} + d_{prepost}) - t_{avg}$. The constant a adjusts the maximum changes to delays, while b, determines the steepness of the function. The output of this function is the absolute change of the delay rather than

a change relative to some maximum value as is the case for the STDP mechanism described in section 2.3.2. This is done in order to maintain interpretability of the functions output. It is important that the constants is chosen so that the function fulfills the constraints given in equation 4.2,

$$\nexists \Delta t_{dist} \in \mathbb{R}, |F(\Delta t_{dist})| > |\Delta t_{dist}| \tag{4.2}$$

which states that there does not exist a value for Δt_{dist} which results in $|F(\Delta t_{dist})|$ being larger than $|\Delta t_{dist}|$. With $|F(\Delta t_{dist}| > |\Delta t_{dist}|$, the delay change would push the arrival time past the average spike time.

Figure 4.4 shows equation 4.1, with constants chosen so that the function obeys the constraints from equation 4.2. This is clear from the plot, as the function only crosses the diagonal at the origin. The corresponding values on the y-axis is therefore never greater than the x-values.

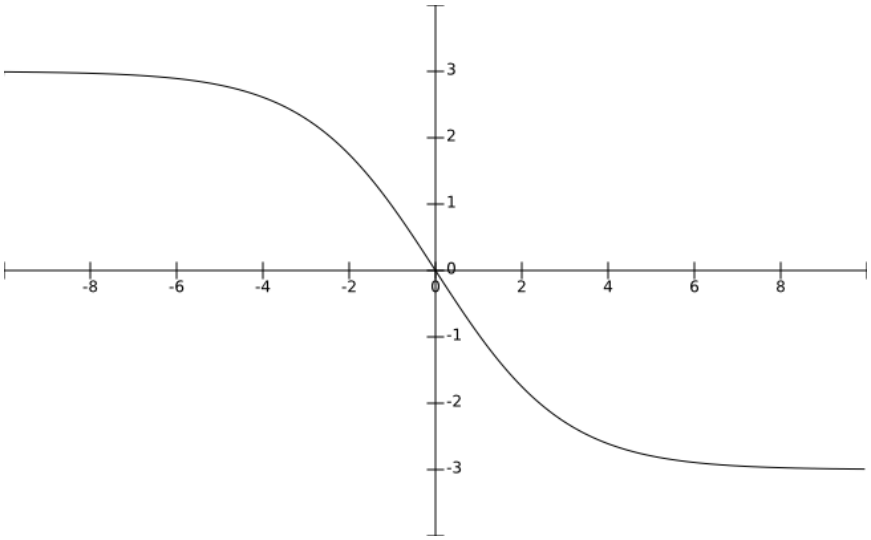

Figure 4.4: Function 4.1, with $a = 6$ and $b = 1.5$. These constant values ensure that the function obeys the constraints from 4.2.

With the mechanism for changing delays formulated, the scope of pre-synaptic spike times that are causally related to the post-synaptic spike must be determined. Observations made in section 3.4.2 shows that the membrane potential returns to within 0.1 mV of its equilibrium state following an input weight of 16 after approximately 11.1 ms. This gives good indication of the extent of the causal relationship between pre- and post-synaptic spikes. Testing with $dt = 0.1$ ms showed that there is a somewhat linear relationship between the weight of the connection and the time window in which a pre-synaptic spike significantly affects the post-synaptic membrane potential. When $w = 16$, the time window is approximately 10 ms, while $w = 13$ gives a window of 7 ms. The linearity breaks down for lower values of w where the time window is in the 6 ms to 7 ms range. For simplicity's sake, it is logical to limit the function to the range $\Delta t = [-10, 0]$ ms for all values of w, but future work could incorporate this into the learning mechanism. It is important to note the

distinction between Δt_{dist} and Δt as $\Delta t = (t_{pre} + d_{prepost}) - t_{post}$, which is the time difference between spike arrival time and post-synaptic spike time.

The learning method has so far only considered negative values of Δt, and does not apply for spike arrival times that occur after the post-synaptic spike. These spikes have no causal relationship with the post-synaptic spike, and one solution would be to apply no delay changes. However, one of the main advantages of physical implementations of SNN's is their efficiency compared to conventional neural networks. It would therefore be more beneficial to try to ensure that as many spike as possible are part of some polychronous pattern, and avoid noisy spikes from interfering with existing spike times. A method could therefore be devised that pushed these noisy spikes away from the post-synaptic spike time. A similar function to the one applied to negative values of Δt can be applied as seen in equation 4.3.

$$G(\Delta t) = -\frac{a}{2} tanh(\frac{\Delta t - c}{2b}) + d \tag{4.3}$$

Choosing the appropriate values for a, b, c and d, one achieves a function that saturates at a positive delay change of 3 for values close to $\Delta t = 0$ ms, and no delay change for values close to $\Delta t = 7$ ms. This function can be seen in figure 4.5. Thorough testing would be needed to find the best suited

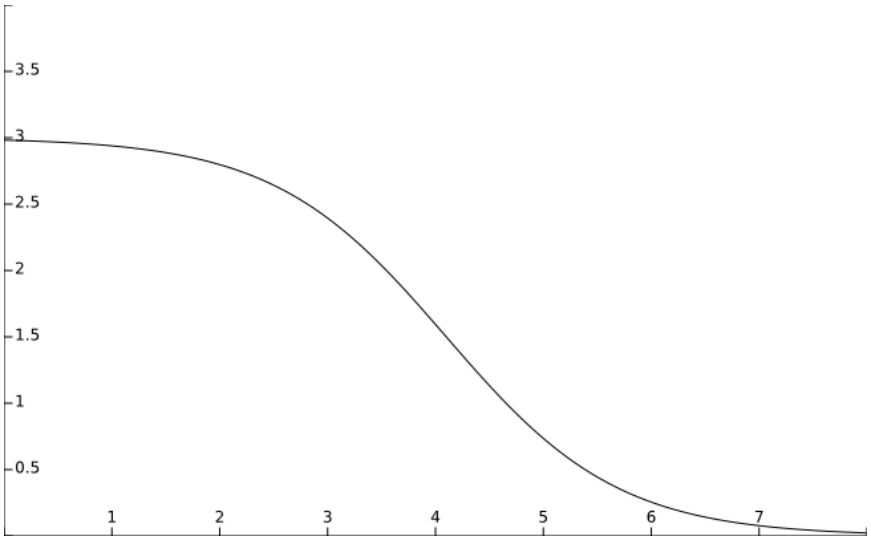

Figure 4.5: The learning function for positive values of Δt with $a = 3$, $b = 0.8$, $c = 4.1$ and $d = 1.5$

time window for activating this part of the delay learning mechanism, so a preliminary window is set to $\Delta t = [0, 7]$ ms.

A complete formulation of the learning rule can therefore be represented by equation 4.4, where ϕ_{low} and ϕ_{high}, are the are the minimum and maximum values of the learning mechanism windows, set to -10 ms and 7

ms respectively.

$$\Delta d = \begin{cases} F(\Delta t_{dist}) = -3tanh(\frac{\Delta t_{dist}}{3}), & \text{if } \phi_{low} \geq \Delta t \leq 0 \\ G(\Delta t) = \frac{3}{2}tanh(2.5625 - 0.625\Delta t) + 1.5, & \text{if } 0 < \Delta t \leq \phi_{high} \end{cases} \tag{4.4}$$

If there are multiple spikes from a single synapse that theoretically triggers both learning mechanisms, the $F(\Delta t_{dist})$ takes precedence. If there are multiple spikes that trigger $F(\Delta t_{dist})$, it is the values with the lowest Δt, that takes precedence.

Figure 4.6 shows an example network with the learning mechanism implemented. It consists of three input neurons connected to a single RS neuron. In addition to these three inputs, there is a fourth input neuron added to illustrate the learning method for positive values of Δt. The black vertical lines of the top plot signifies spike times, while the colored vertical lines gives the corresponding spike arrival time at the RS neuron. Similar colors in the raster and delay plots indicates information related to the same neuron. As the figure shows, the three first spike arrival times are within the time window that will trigger the learning function for $\Delta t \leq 0$ ms. The learning method will therefor adjust the delays so that spike arrival times are shifted toward the average spike arrival time, which is likely in between 7 ms and 8 ms. The connection of the first input neuron should therefore experience a noticeable decrease in delay. The second input neuron will experience an increase in delay of a similar magnitude. Finally the third input neuron will experience a slight decrease in delay. The middle plot of 4.6, showing the corresponding delays as functions of time, confirms these observations. A second spike was added to input neuron 3, to illustrate that spike arrival times prior to the post-synaptic spike times have precedence, when triggering a learning mechanism even though the second spike arrival time is closer to the post-synaptic spike time. As mentioned, the spike arrival time of the fourth input neuron is past the spike time of the RS neuron, and therefore experiences an increase to its delay.

Using the new delays and an identical input spike pattern, the second set of plots in figure 4.6 shows that the spike arrival times are now much more aligned resulting in a decrease in spike time at the RS neuron.

Since the training of the delays is an iterative process, the delays for input 1 and 2 continues to experience a slight shift in order for the spike times to align even better. Once all three spikes are perfectly aligned, the delays no longer experience shifts in this configuration. The spike arrival time of the fourth neuron is now pushed passed the post-spike window of the RS neuron, and therefore no longer experiences changes to its delay.

This learning method also solves the issue that occurs when the average spike arrival time is before the minimum spike arrival time of one of the contributing axons. As figure 4.7 shows, the average spike arrival time for the three input neurons is behind the spike time of the third input neuron. For the first iteration, this neuron experiences a decrease in delay, but since the delay can not be negative, the minimum spike arrival time for this neuron is $t_3 + d_min$ where d_min is the minimum delay which is set equal to

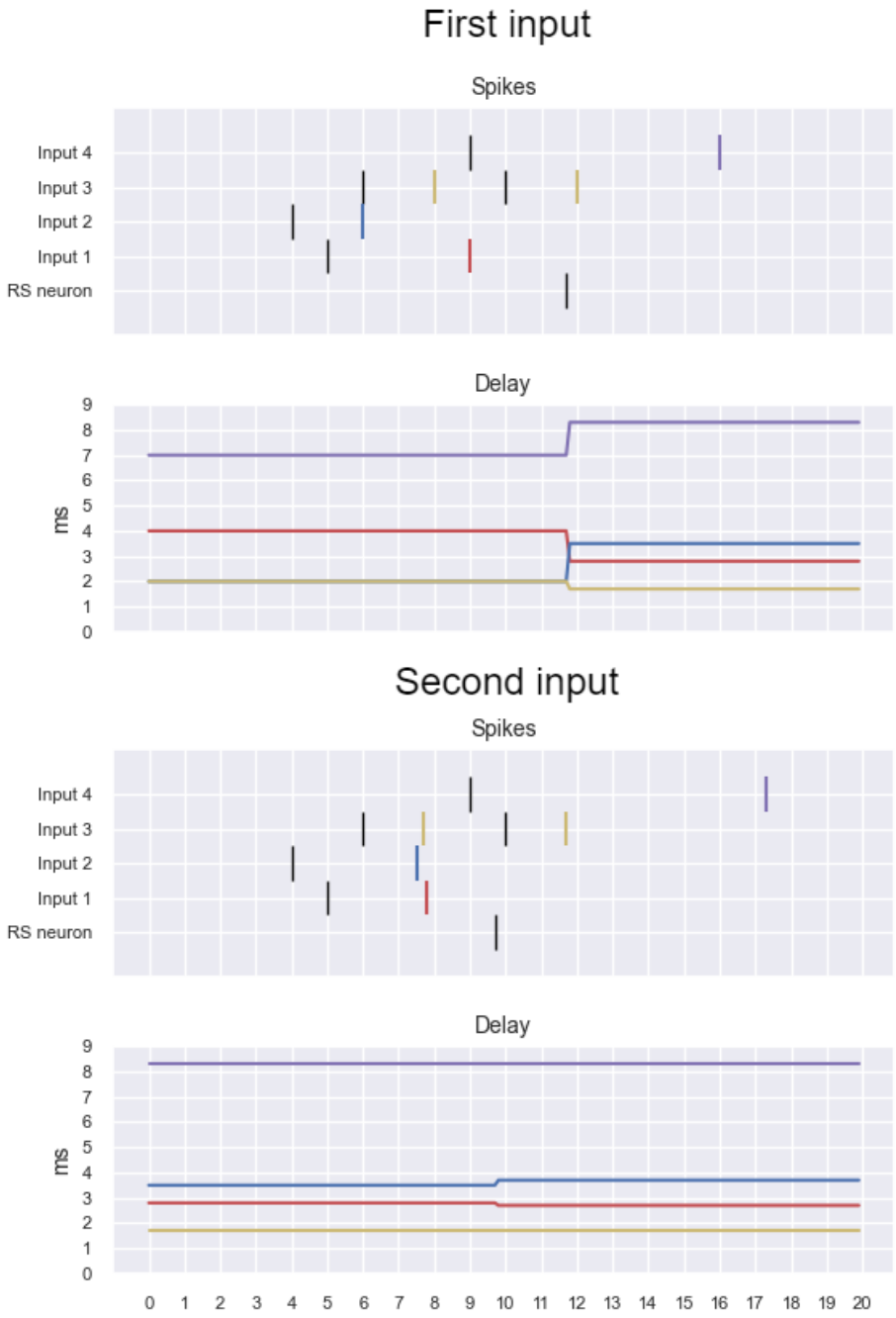

Figure 4.6: Two identical inputs are given, the top set of plots show the initial activity and delays, while the second set shows the same input but with the new delays.

the dt-value of the simulation, and is therefor 0.1 ms in this case. However, since the spike arrival time of input neuron is now kept constant at its minimum value, the other two spike arrival times will slowly be shifted towards it, since the average spike arrival time will always be somewhere between between the three. After some iterations, continuously updating

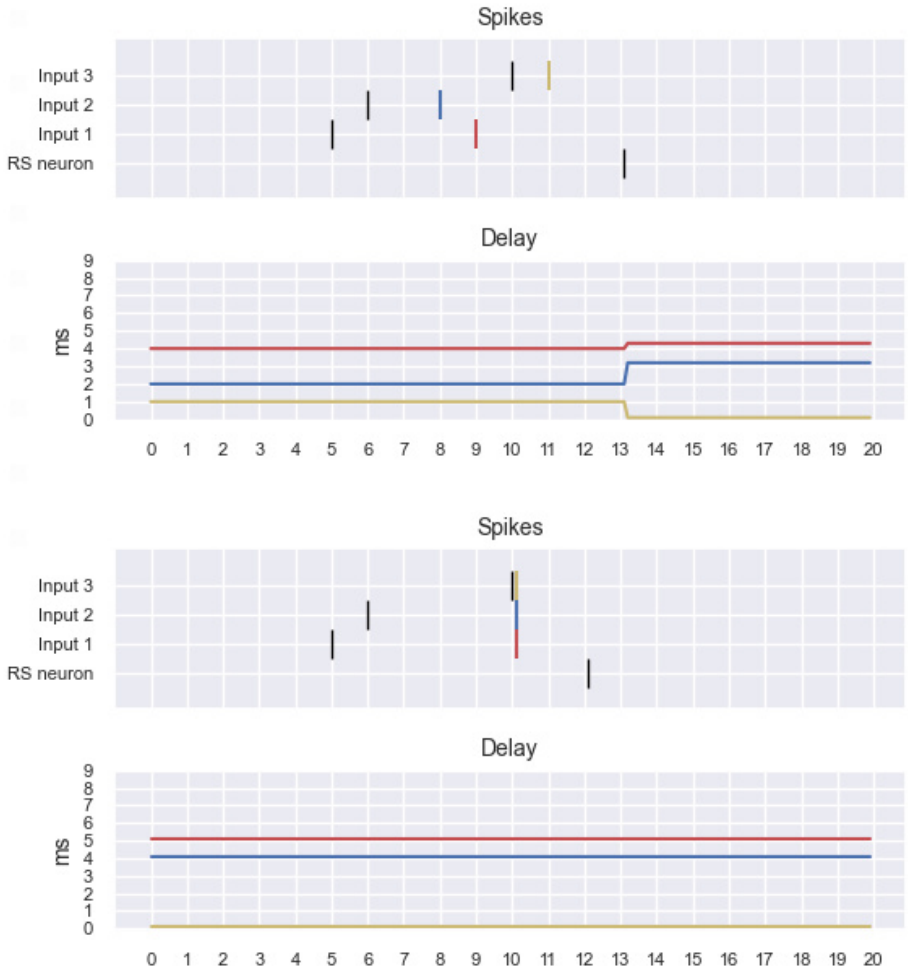

Figure 4.7: The spike arrival times align even though one of the spike times is after the average spike arrival time of the first iteration.

the delays to their new values, which is equivalent to inputting a periodic pattern, the bottom plot of figure 4.7 shows that they eventually align to the spike arrival time of input neuron 3.

4.3 Categorizing delay behavior for networks exposed to RSL and rate-coded input

The behavior of the network and its ability to learn, and consequently perform classification, depends on the behavior of the delays. In this section two simple topologies were tested, consisting of two inputs connected to a single RS neuron, and three inputs connected to a single RS neuron. These two topologies was exposed to three different types of inputs and the delays of the network was categorized based on their behavior. The purpose of categorizing the delay behavior for the different

types of input is to understand how these inputs affect the delays. As the final set of delays of a network after training can be considered the solution of the training period, the desired behavior of the delays is for them to converge. This would indicate that the network has settled on some optimal state which accommodates for all the inputs that the network has been exposed to.

4.3.1 Input patterns

In order to understand the effect of the input pattern on the network, several instances of the three input types was generated and tested on networks of various delays. The three distinct types of input patterns that was tested can be seen in figure 4.8.

The simplest of these pattern types is the RSL pattern, which consist of a set of spikes which have a constant time shift between them, and this pattern is repeated with a certain interval. The purpose of this pattern is to see how the network adjusts to a constant repeating pattern. Preliminary results showed that RSL inputs results in convergence as long as the input causes activity in the post-synaptic neuron. Therefore, this input type was not examined further in this context.

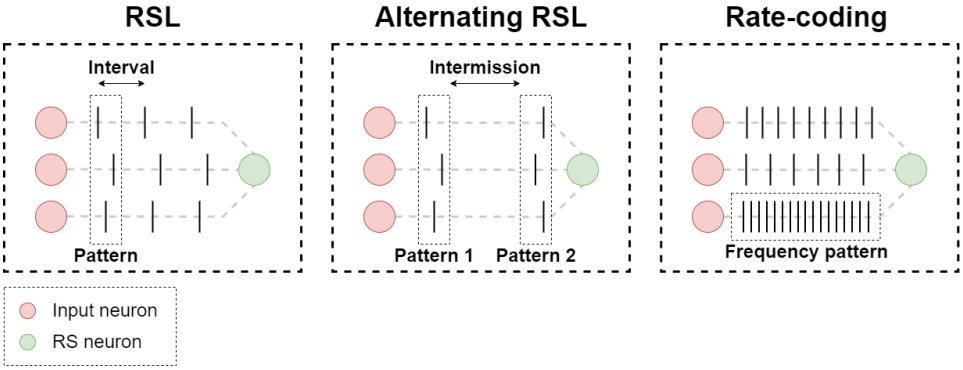

Figure 4.8: Three simple types of spiking inputs. The leftmost figure shows a RSL pattern consisting of a pair of three spikes with a specific relative offset, repeated with a specified interval. The middle figure shows an alternating RSL pattern consisting of two patterns, similar to the the leftmost figure, but alternating between the two with a specified intermission period between then. The final figure shows an input type where each input neuron fires with a specified spike interval.

The second input type is the alternating RSL pattern. This pattern is generated in a similar way as the RSL pattern, except it consists of two different sets of time shifts, and they are alternated with a longer intermission between them. This intermission is added so that the network can return to equilibrium before the new pattern is introduced. The purpose of this pattern is to see how the network adjusts to two alternating input patterns.

The last input type is the rate-coded pattern and it differs from the

previous two in a fundamental way. The set of spikes that constitutes the two previous patterns are time locked. In other words, the time shift between the set of spikes is always constant. For the rate-coded pattern, each neuron fires with a specific firing rate, and therefore the offset between the firing times of the input neurons is not necessarily constant, unless the firing rate of one input is a multiple of another.

Due to the vast number of parameters that determine the configuration of even small networks of a few neurons, it is not feasible to do a thorough test of all such configurations. The parameters for these configurations was therefore kept within logical intervals according to computational constraints.

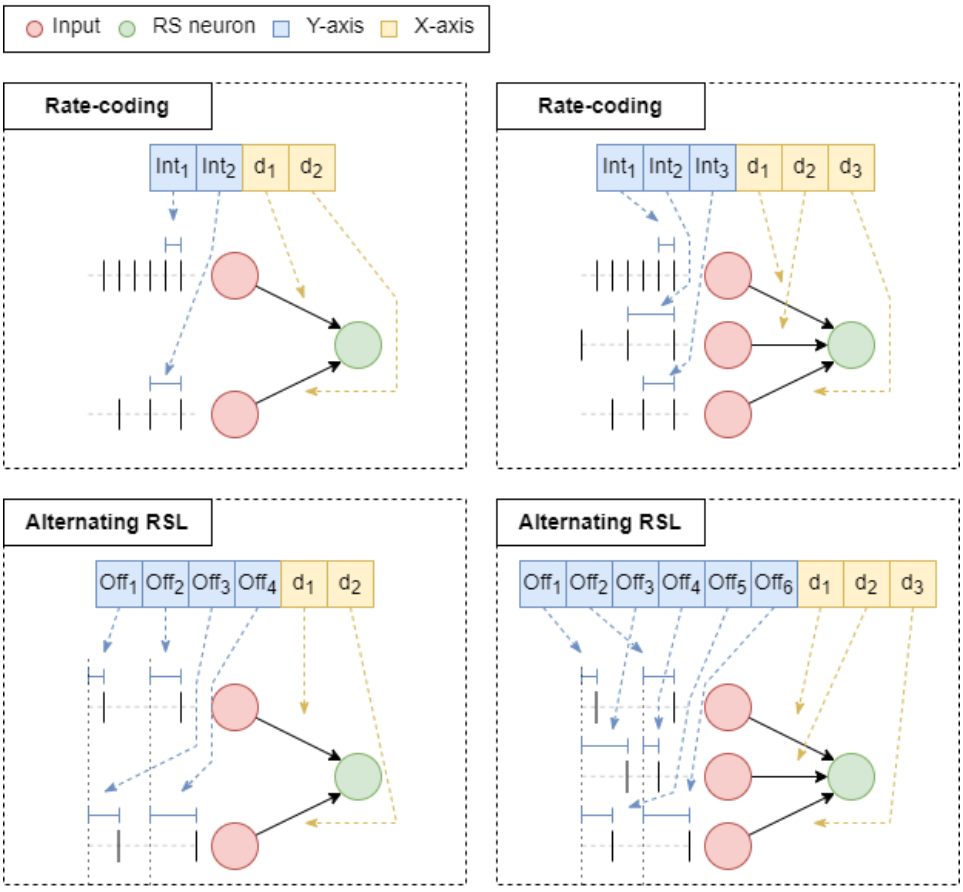

Figure 4.9: There are two network topologies, each subject to two different inputs. The parameters for the testing therefore consist of the delay of the network and parameters related to the input. For the rate-coded input it is the interval between spikes for each input. For the alternating RSL input it is the offset from the pattern start for both patterns in each input.

With two topologies subject to the two input types, alternating RSL and rate-coding, a total of four main configurations were tested with various parameters. Figure 4.9 shows the four configurations along with their set of parameters. The parameters are color coded based on what axis they are located on in the plots that show delay categories later in this section.

A common parameter for all configurations is the delays, although the intervals vary due to computational constraints. Arrows show the physical meaning of each parameter.

For the rate-coded input, the Int_i or *Interval*, determines the time between spikes for each input neuron. Although rate-coded input is typically parameterized by frequency, the interval was used in this case since it is easier to interpret the interval between spikes instead of frequency in this context. For the alternating RSL input, the Off_i or *Offset*, determines the time from the start of the pattern to the spike of each input. This initial pattern is then repeated every 30 ms for 500 ms, followed by a 500 ms dormant period. The new pattern is then introduced in a similar manner. The two patterns alternate in this fashion for the duration of the simulation.

Including parameters for both the inputs and delays, gives an understanding of the significance of the delay initialization and the input instance on the behavior of the delays.

All simulations described in this section were run for 10000 ms, which was deemed sufficient for the delays to settle into a category.

4.3.2 Categorization method

The delays of the networks can be expressed as functions of time, and when viewed in this manner the behavior of the delays can be separated into specific categories. Seven main categories have been identified and include:

- Converging
- Repeating
- Increasing
- Decreasing
- Maximum
- Minimum
- Dormant
- Uncategorized

The *converging* category is the desired behavior of the delays, as it implies that the delay has converged to a specific value that accommodates for the network activity. This category is given when the standard deviation, $\sigma_{d(t_{int})}$, of a specified time interval, t_{int}, of the latter part of the delay function is within a specified threshold, σ_{th}. The delay behavior of a connection is therefore categorized as converging when $\sigma_{d(t_{int})} < \sigma_{th}$. σ_{th} was set to 0.1 and t_{int} was set to 2000 ms. The selection of the time interval was based on observations that repeating delay patterns could have quite long periods with unchanging delays which would otherwise be categorized as converging. The length of these unchanging periods is largely due to the way the input is generated.

The *repeating* category is a special case where the delay function expresses a periodic repeating behavioral pattern. The categorization is ideally determined by taking a similar time interval as for the *converging* pattern, and calculating the correlation coefficients of this time window with each previous time shifted window of equal length. If the correlation exceeds a specific threshold, the *repeating* category is assigned. This method allows for some leniency when it comes to matching the repeating pattern with the previous delay function. However, this is a computationally unviable option as there is a high number of possible time shifted windows. Instead, a direct string comparison is done, which requires a complete match of the entire chosen time window of the delay function. This time-window is also set to 2000 ms.

The *increasing*, *decreasing*, *maximum* and *minimum* categories all describe diverging delay patterns, with the latter two pertaining to delay values that have saturated toward the maximum and minimum values allowed, respectively. These four categories described undesired behavior where the delays are either not converging, or is limited by the bounds set for the simulations. Identifying increasing or decreasing delays is done by doing linear regression on the delay function and then retrieving the slope of the resulting line. If the absolute value of the slope exceeds a specified threshold, set to 0.001, the delay is either increasing or decreasing, which is based on the sign of the slope. Delay saturation is determined by calculating the mean delay value of a time interval, set to 1000 ms, and checking if it is within a certain margin, set to the same threshold as for the converging method, of either the maximum or minimum allowed delay values. The choice for a shorter time interval is because the delays require more time to saturate, and because once the delay has reached the maximum or minimum values it is unlikely that it will decrease again.

Due to the logic of the converging category, low-activity interactions where there is little to no change in the delays could potentially lead to miscategorization. To avoid this, another category is introduced that captures dormant neuron interactions. This category is triggered when activity is less or equal to 0.1 $^{spikes}/_{s}$, which was observed to be the approximate threshold for achieving any meaningful changes to delays.

The categories mentioned so far are determined through various means, but does not capture all behavior, and the last category is therefor named *uncategorized*. As the name suggests, this label is given when none of the other categories apply. The logic for categorizing individual delays can be seen below.

The specific order of categorization is important. Initially it is determined whether the network is dormant. The next category that is checked for is the saturated category. The reason for this is that when a delay is saturated, it is often constant, which could easily be miscategorized as being a converging delay. Other times the delays behave in a repeating manner, similar to that of a repeating delay, only it is situated around the delay limits. It is therefore important to first test for saturation to avoid misclassification. Then the converging category can be tested for, followed by the repeating category. It is important that the converging category

if *dormant* **then**
| Categorize as dormant;
else if *saturated* **then**
| Categorize as saturated;
else if *converging* **then**
| Categorize as converging;
else if *repeating* **then**
| Categorize as repeating;
else if *diverging* **then**
| Categorize as diverging;
else
| Categorize as uncategorized;
end

Algorithm 2: Logic for categorizing individual delays

is check first, as a converging delay is constant and would therefore fall under the repeating categorization logic. Finally the diverging category is checked, and if none of the categories are given, the delay is labeled uncategorized.

The purpose of categorizing the delay behaviors is to understand the effects of a specific input or delay initialization on the network. When considering that there are eight delay categories, then for a simple network of 2 connections, the number of possible combinations when order matters is 64. It is theoretically possible for a delay function to be within more than one category. For example, it can be both increasing and saturating at the maximum allowed delay value, further increasing the number of categories. Having this many categories for each network is impractical and the number of categories needs to be compressed.

if *dormant* $\in$ *categories* **then**
| Categorize network as dormant;
else if *increasing or decreasing or maximum or minimum* $\in$ *categories* **then**
| Categorize network as diverging;
else if *uncategorized* $\in$ *categories* **then**
| Categorize network as uncategorized;
else if *repeating* $\in$ *categories* **then**
| Categorize network as repeating;
else
| Categorize network as converging;
end

Algorithm 3: Logic for categorizing networks of a single RS neuron.

Similarly as to the categorization of individual delays, the order in which the network categorizing logic is applied is important. Initially, the activity of the RS neuron is checked, and if the spike rate is below the specified threshold, the network is classified as dormant. The next step is to check for divergence. One can consider *increasing*, *decreasing*, *minimum* and

maximum as diverging. It is unwanted behavior, and does not necessarily need be described in more detail in an overview of network activity. One can also assume that any connections that share the same post-synaptic neuron, have delay functions that are to some degree correlated, as a change in one will likely lead to a change in the other. It is therefore logical to say that if one connection is diverging, the connection pair have not converged, and the pair can therefore be said to be diverging. If there are no diverging connections in a pair, but at least one is uncategorized, the pair can be said to be uncategorized. The reasoning behind this is that the uncategorized connection could actually be diverging which takes precedence over the other categories, and the pair can therefore not with any certainty be put in any of the other categories. If none of the mentioned categories are present, but at least one repeating delay function is observed, then the pair can be said to be repeating. This is the case because the final category is the converging category, and it is logical to assume that all connections in a pair must be converging for them to be considered in this category, since any other behavior means the pair has not settled. If none of the other logic has been triggered, the network can be considered to be converging. This logic can be seen in algorithm 3.

This results in a set of five categories that describe each connection pair, which is more practical for making assumptions and observations on entire networks.

Figure 4.10 shows one example from each network delay category. The legend indicated what color corresponds to what connection. The RS neuron is always 0, while the inputs are numbered from 1 to 3. In the dormant example, the RS neuron is dormant, and therefore none of the connections experience any change. The diverging example consist of one *uncategorized* delay from input 1, one *increasing* delay from input 2 and one delay saturated at the minimum value from input 3. Since the network consist of at least one diverging connection, the network is categorized as diverging.

The repeating example contains two repeating connections from input 1 and 2, and one converging connection from input 3. Since the repeating connections takes precedence, the network is labeled repeating.

The converging example contains three converging delays. The connection from input 1 and 2 overlap as they have the same spiking parameters.

Finally the *uncategorized* example consist of two *uncategorized* delays. Both delays are increasing and decreasing, the sum of which does not activate either of those categories. Nor does it activate any of the other categories and is therefore given the *uncategorized* label.

4.3.3 Rate-coded input

For the first test configuration, various versions of the rate-coded input described in section 4.3.1 are explored. Starting with the simplest network that is of interest, which consists of 1 RS neuron and 2 inputs, with connection weights of 16. This leads to two individual connections

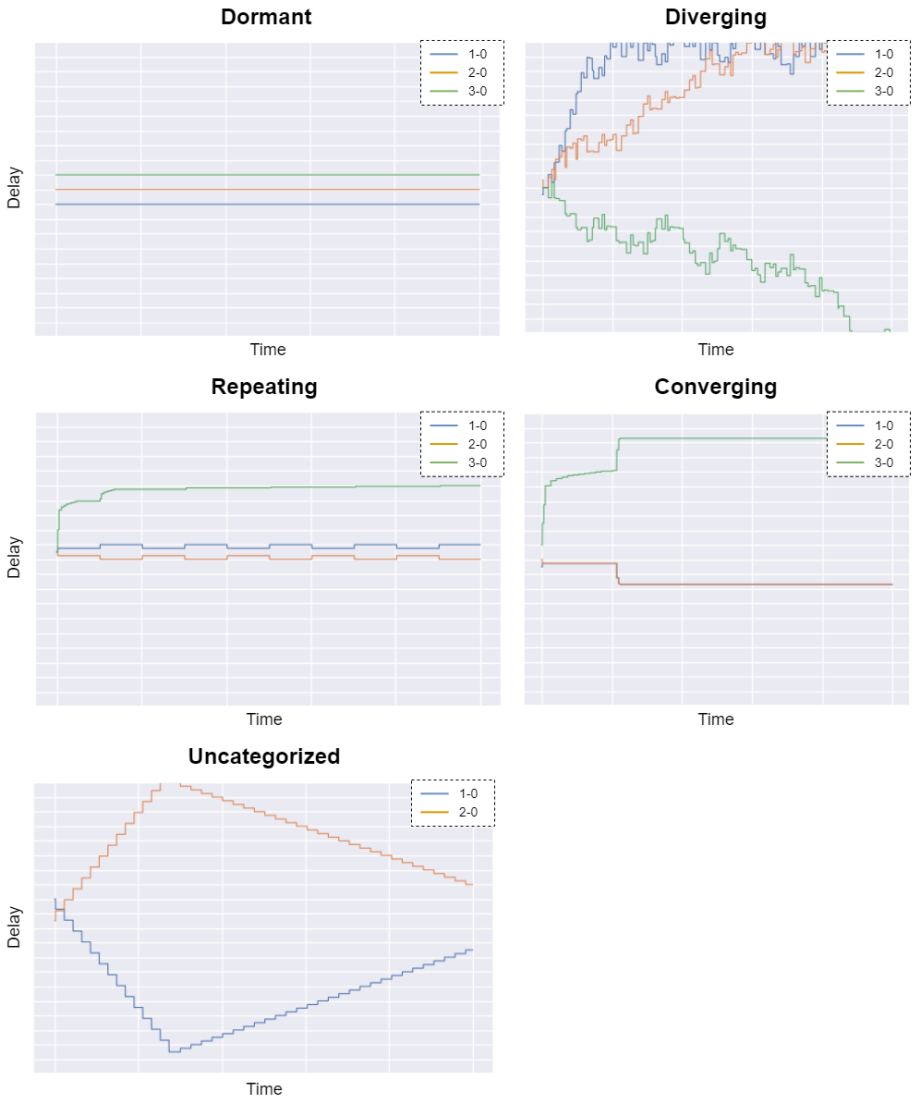

Figure 4.10: Shows an example from each network delay category.

connecting the two inputs to the RS neuron. Each connection is initialized with a delay chosen from the interval [10, 30] ms. The limits for the delay values in the simulation is 0.1 ms and 40 ms. The chosen interval for initialization will therefore allow for the most changes in each direction, reducing biased saturation to either extreme. Each input is producing a spike train with an inter-spike interval chosen from the interval [20-30] ms. The aforementioned parameters results in a set of 48 400 unique simulation configurations.

The top plot of figure 4.11 shows the categories resulting from the logic described in section 4.3.2, for each simulation with parameters determined by the x- and y-axis of the plot. The x-axis shows the pair of delays with which the models were initialized, and the y-axis shows the pair of spike intervals for the two inputs. Both axis are sorted low to high based on the differences in the parameter values. The reason for the sorting method is

that it is hypothesised that the delay behavior is more dependent on the difference in the parameters than the absolute values.

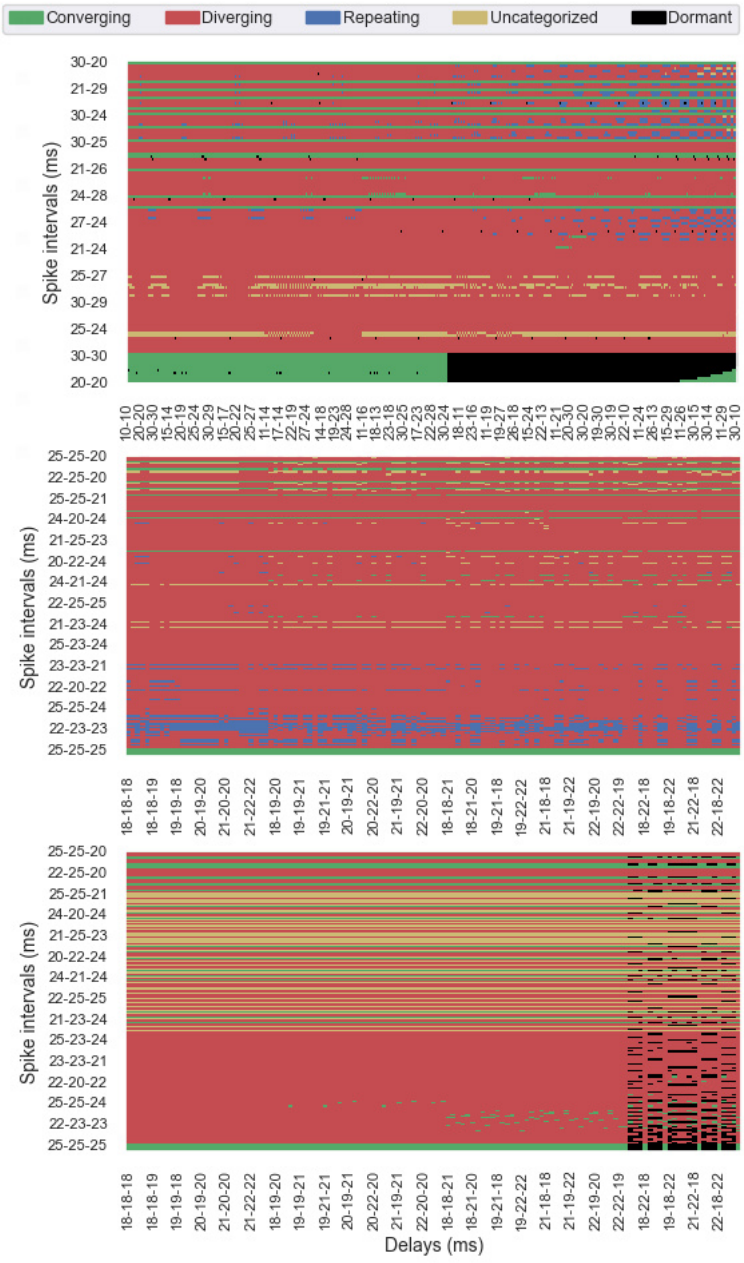

Figure 4.11: Top: Delay categories for two inputs and one neuron. Middle: Delay categories for three inputs and one neuron. Bottom: Delay categories for three inputs and one neuron connected with a weight of 8.

The plot clearly shows horizontal trends indicating the the delay behavior is more reliant on the input activity rather than the delay initialization. The most prominent category by far is the *diverging* category. A high degree of diverging delays is typical for an input pattern where the spikes from each input neuron is not time-locked, or in other words, the

offset between pairs of input spikes are not constant.

There is also a notable presence of both *converging* and *dormant* delays. The convergence is largely congregated around low delay differences and equal spike intervals. There is also occurrences of convergence at higher spike interval differences, which also seem less affected by the delays. It is possible that this convergence stems from pairs of spike intervals that periodically align.

There is a clear trend for repeating delays at the highest differences in both spike intervals and delay pairs. It seems that the repeating categories are more dependent on the initialized delays as can be seen by the dotted pattern. It is likely that specific combinations of spike intervals and delays lead to the equivalency of an alternating RSL input.

The top plot of figure 4.12 shows the spike rate of the RS neuron based on the same network configurations. There are clear trends both horizontally and vertically. High spiking activity seems to be centered around identical spike intervals on the y-axis and centered around similar delays on the x-axis. Interestingly, there is also high activity when the delays are very offset as the previous input spike from one input neuron suddenly lines up with the current spike from the other input. It is also clear that the areas with high activity strongly correlate with the convergent category, which is logical as convergent delays will likely match well leading to a high spike rate.

The next step towards more complex networks is to include another input. In similar fashion as the previous configuration, the three inputs are connected to the RS neuron. The connection strength is kept the same, at 16. Due to computational restraints and observations from the previous results, the intervals governing the possible simulation configurations are reduced. The range for input spiking intervals is set to [20, 25] ms and the possible values for delays are from [18, 22] ms, again centered around the middle of the allowed delay range. With a selection of three values from each set, the number of possible combinations amount to 27 000.

The delay categories of this configuration can be seen in the middle plot of figure 4.11. This topology shows perhaps a bit more dependence on the delay initialization while having less clear trends regarding clustering delay categories to either ends of the plot. The diverging category is still significantly more common than the other categories.

Again the converging delays are most prevalent where the spike intervals are equal. The repeating delays appears close to where the difference in spike intervals are lowest, but seems dependent on the delays as there is a more sporadic pattern.

The middle plot of figure 4.12 shows the spike rates of this topology. It generally shows high activity when the spike intervals are equal. The exception seems to be when non of the delays are equal. There is then a general trend of lower activity as the spike interval differences increase.

The previous networks rely on connections with a weight of 16, and therefore only two spikes are required to align for a post-synaptic spike. If the weight is reduced to 8, three concurrent spikes are required to elicit a post-synaptic spike. The bottom plot of figure 4.11 shows the resulting

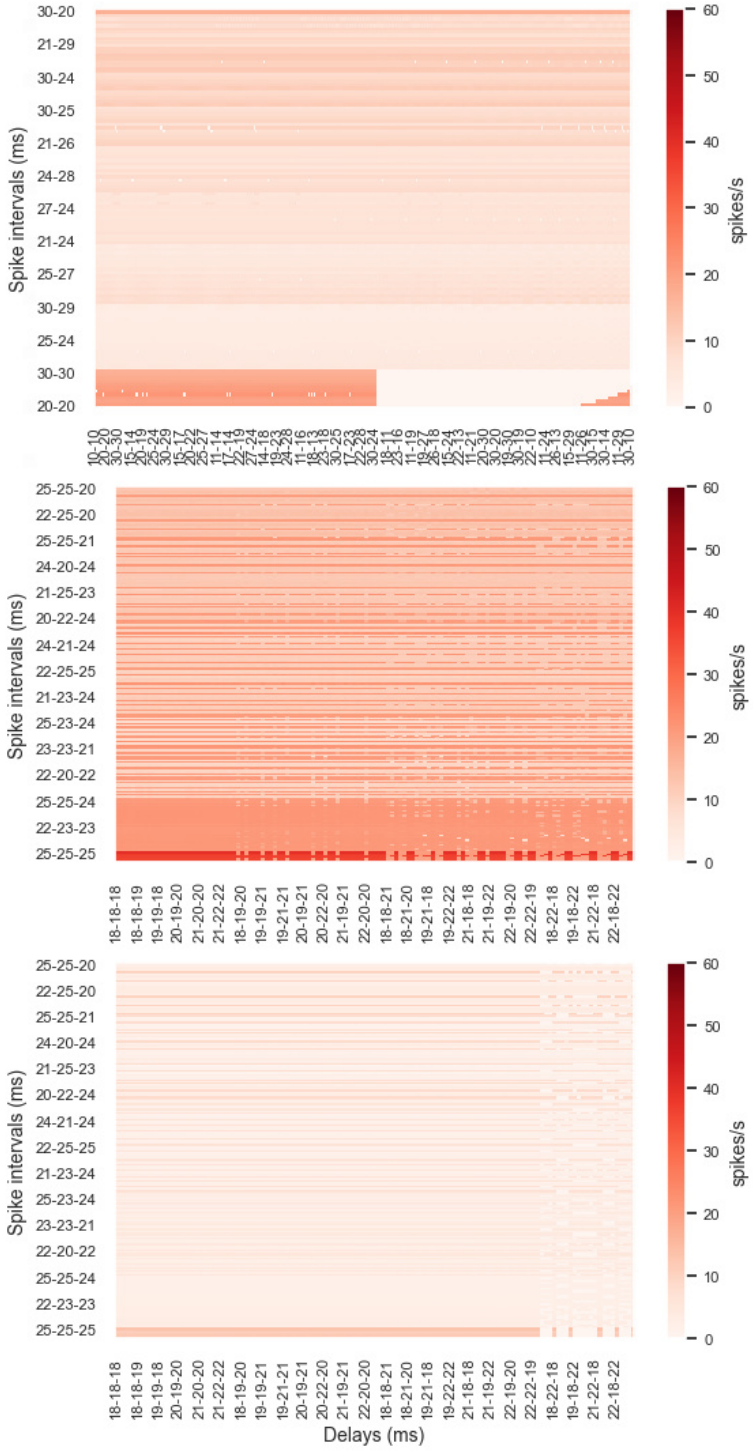

Figure 4.12: Top: Spike-rate plot for two inputs and one neuron. Middle: Spike-rate plot for three inputs and one neuron. Bottom: Spike-rate plot for three inputs and one neuron connected with a weight of 8.

delay categories of this scenario.

This reduction in connection strength results in dramatic changes to the

delay categorization plot. The most obvious is perhaps the emergence of a field of dormant connections in the right part of the plot, which seems dependent on both the delays and spike intervals. There also appears to be more uncategorized behavior.

The corresponding spike-rate plot can be seen at the bottom of figure 4.12, and is similar to that of the previous network type, but with a significantly lower level of activity.

4.3.4 Alternating RSL input

The other input type that is of interest is the alternating RSL input. For the simplest network of 1 RS neuron and two input neurons, the configuration consists of 4 spike offsets and 2 delays. The two first offsets constitutes the first pattern. Each of the two numbers describes the spike time chosen from [0,5] ms for each of the two input neurons. These spike times are then repeated every 30 ms for 500 ms. This is followed by a dormant period of 500 ms before the other spike pattern is introduced in a similar manner as the previous pattern. This process is repeated for the duration of the simulation. Since the input now has more parameters it was necessary to reduce the interval for the delays to the range [18, 22] ms.

As expected, the top plot of figure 4.13 clearly shows that the primary category for this type of input is repeating. The second category that is present is the converging category. The converging delays are most prevalent for higher difference in spike offsets, while neither categories seems very affected by the delays. The other three categories are practically non-existent.

According to the top plot of figure 4.14, the spiking activity is quite evenly dispersed with the only trend being slightly lower activity for high differences in spike offsets.

There is an interesting change when the networks size is increased to include a third input neuron as shown by the middle plot of figure 4.13. There is no longer an obvious preferred category, and there is no longer any recognizable trends. The only two categories that does not seem to be present is the uncategorized and dormant categories. Again, the corresponding spike-plot seen in the middle of figure 4.14 shows higher activity trends when differences in spike offsets are low.

When the weight between the three inputs and the RS neuron is reduced to 8, the delay category plot, seen at the bottom of figure 4.13, looks quite similar to the delay plot seen for the smaller network. There is a higher trend of convergence and some areas of dormancy. The corresponding spike-rate plot, seen at the bottom of figure 4.14, shows a relatively low level of activity across the plot.

Upon further inspection of a few uncategorized cases, there appeared to be some trends. A typical behavior for the low-activity interactions was that the low spike-rate lead to the delays changing slowly, and consequently not being registered by any of the categorizing methods within the given time frame. Another strange observation was pairs of delays were one increased, while the other decreased until one reached

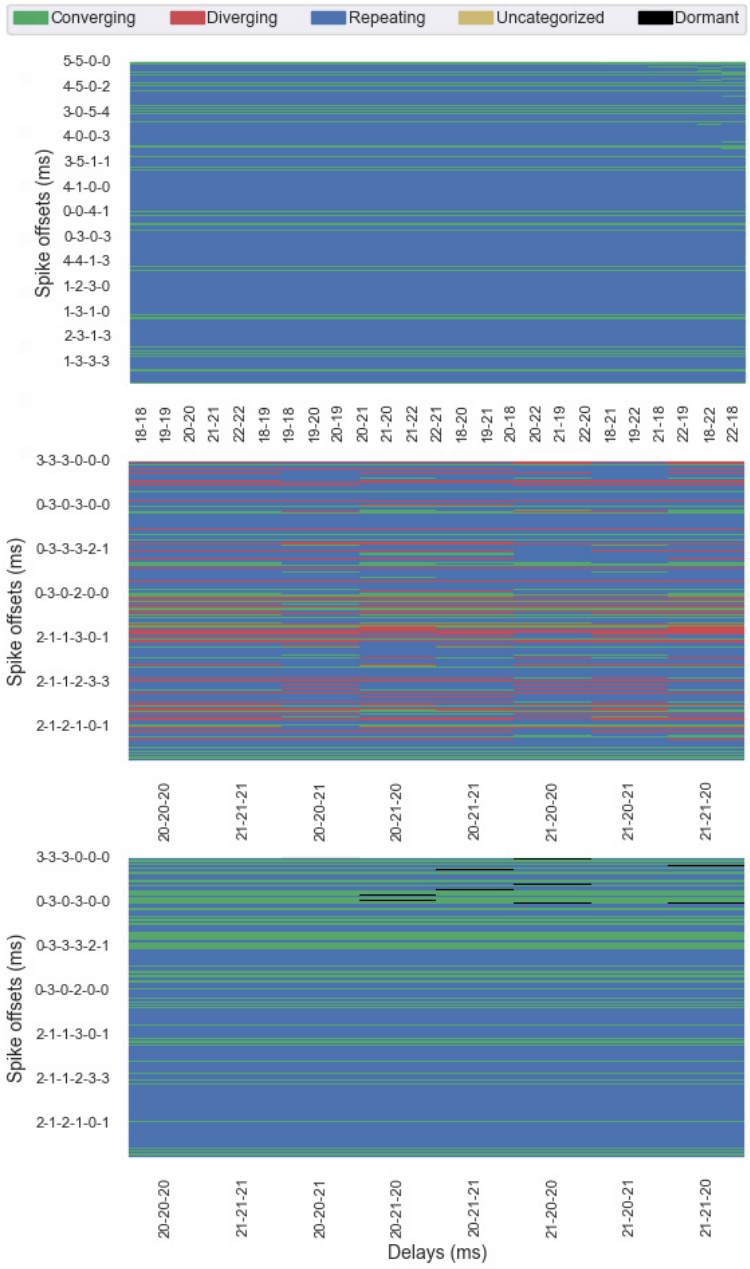

Figure 4.13: Top: Delay categories for two inputs and one neuron. Middle: Delay categories for three inputs and one neuron. Bottom: Delay categories for three inputs and one neuron connected with a weight of 8.

the delay limit, upon which the behavior reversed, and the first decreased while the second increased. This is likely the result of a change in one delay, while the other delay is prohibited from changing by the delay limits, leading to a switch of the spike arrival time, warranting a new direction of delay change. Lastly, a typical observed behavior was divergence to the delay limits, followed by erratic, non-repeating behavior.

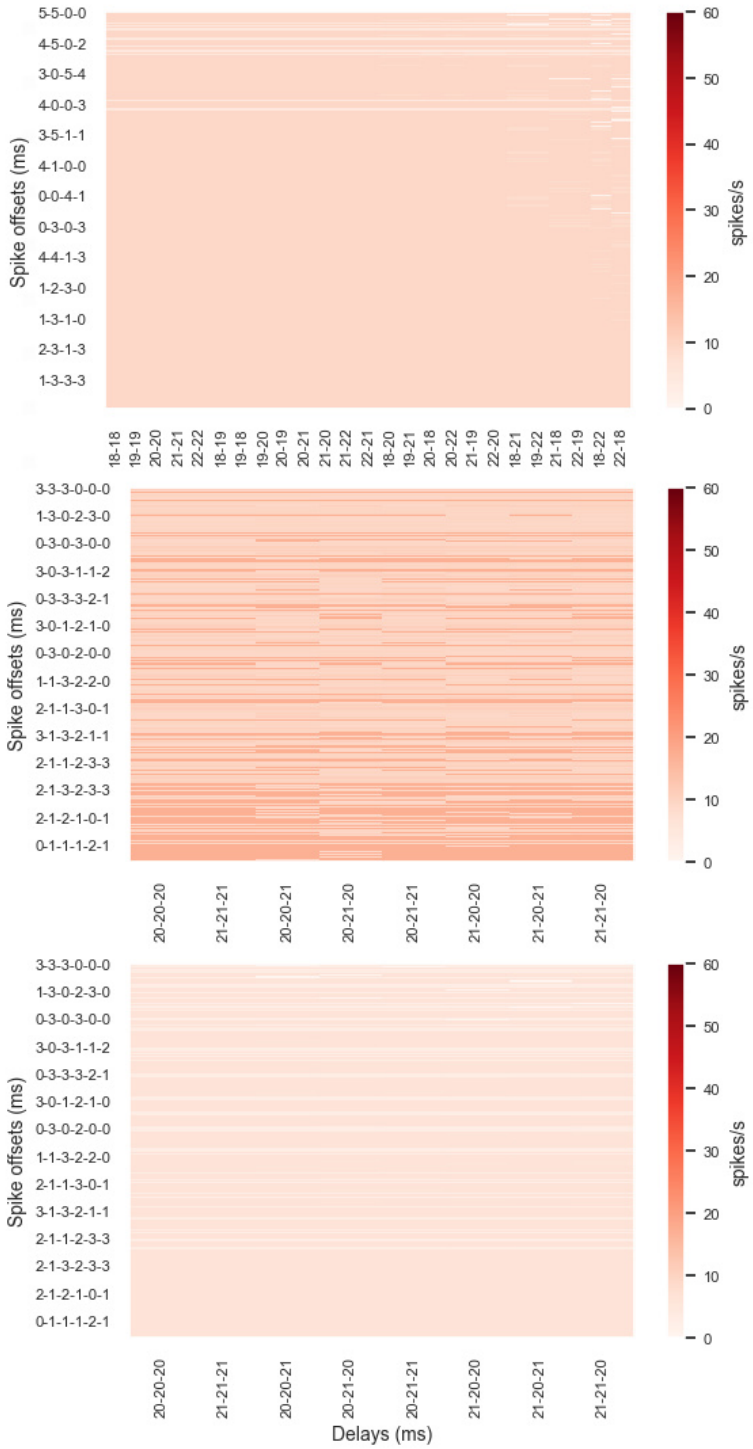

Figure 4.14: Spike-rate plot for networks with three inputs, one RS neuron and weights of 8.

4.3.5 Conclusion

Based on these findings it is clear that the rate-coded pattern in large part leads to diverging delays regardless of the three configurations tested. This

is an indication that this encoding method might not be suited for this learning method. The alternating RSL input leads to mostly repeating delays, which is at least better than converging. It also appears that reducing the connection weight might lead to more convergence.

In general, the highest activity is seen when delays converge. For both input types it appears that reducing the connection weight limits the spiking activity quite significantly. If such low weights are used, it might require higher connectivity. For the remaining sections, the RSL encoding method will be used as it appears to result in more stable delay behavior than the rate-coded input.

Chapter 5

Input classification through polychronous group detection

This section will investigate the performance of the delay learning method by applying it to three different topologies and performing classification of inputs. The three topologies are feed-forward, ring-lattice and reservoir. The reservoir topology can also be referred to as a randomly connected recurrent network.

A simple but effective method for information encoding is to use RSL described in section 2.4.3. Based on the observations of section 4.3.2, this was also found to be a better suited encoding method than rate-coding for the proposed learning mechanism. In order to have a clear definition of what a class would be, an input class is represented by some specific spike order from the input neurons and the instances of that class would consist of any variation that still follows that spike order. This could also be considered a type of ROC method as the two coding schemes are quite similar, however for consistencies sake, the method will be referred to as RSL.

The method is illustrated in figure 5.1, which shows two classes, A and B, and three instances of each class. Even though the spike times differ within a class, the order is the same. Continuously repeating such an input order should theoretically train the network to accommodate for all the variations of the input classes that is exposed to the network.

The three topologies tested in sections 5.3, 5.4 and 5.5, were given 12 instances of one input class, followed by 12 instances of another class. This was done in attempt to see if the delay learning method could adjust for and properly classify one input, and then readjust for a new pattern and correctly classify that as well.

5.1 Polychronous group detection method

When attempting to extract information from a spiking neural network, there are two methods that were considered promising candidates for this project. The first one is to feed the network activity into a regression model, which undergoes supervised training with labeled data. The regression

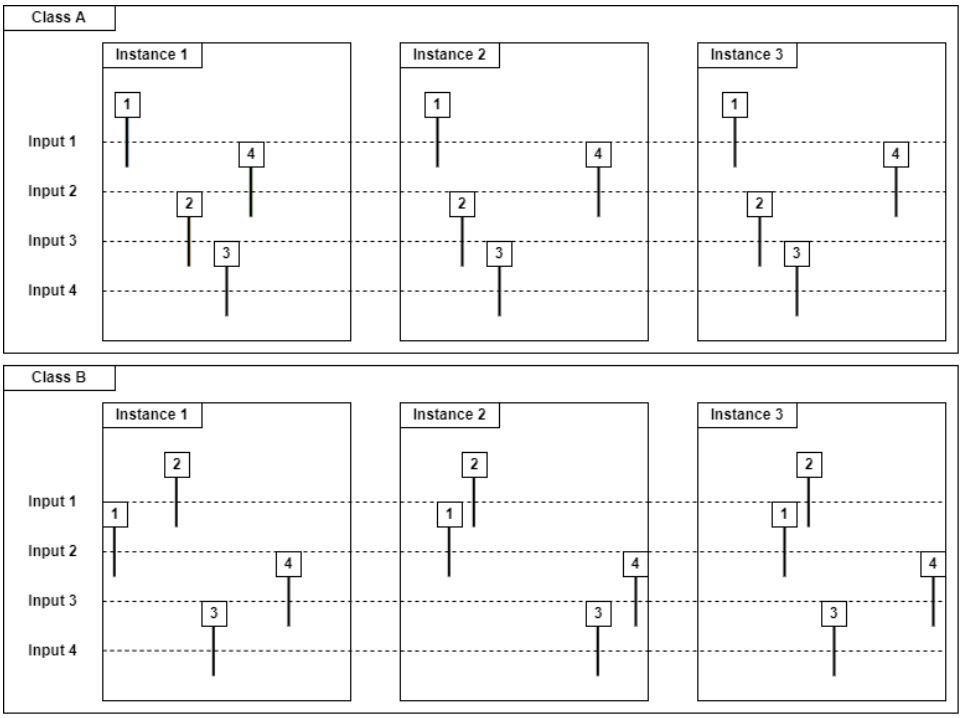

Figure 5.1: Two input classes with three instances of each class.

model then learns what output activity corresponds to the correct input class. This is the most common method used for reservoir computing, but is for obvious reasons less biologically realistic.

Another method that does not rely on another predictive model is to detect PGs. There exists methods for offline detection of PGs, but these methods are often slow. An efficient online method was proposed by Chrol-Cannon et al. [6], where subsequent spikes build up an identifying code. This method is sensitive to the exact order of spike arrivals at the post-synaptic neuron, something which would likely result in mis-identification when delays are adjusted consequently altering spike order. An new algorithm loosely inspired by this method was therefore created.

The idea that governs the proposed method for PG detection, relies on the creation of data structures that can store spiking activity in a meaningful way. The spikes must be stored so that the order of a set of pre-synaptic spikes that cause a spike in a post-synaptic neuron is insignificant. It is, however, important that the overall sequence of spikes is retained, in terms of which neuron causes which neuron to spike. It is important to distinguish between individual spikes of a specific neuron, as one neuron might fire multiple times as a result of one input. Consequently, this neuron would appear multiple places in the PG structure.

The general data structure that represents the PGs starts with a single spike from the input neurons, and then branches out to include all subsequent neurons that spikes as a result of a specific spike by said input. These subsequent spikes are therefore ordered in a hierarchical branching

structure from initiating spike to the final spike in each branch that does not elicit a spike in another neuron.

The method for building the PGs relies on each individual neuron to add its own structure to the structure of each pre-synaptic neuron that contributed to its own spike. The structure that is added is empty at this point, but will be filled by the structures of post-synaptic neurons that spike as a result of this spike. Each neuron has one data structure for each spike it produces, which will contain different substructures depending on the subsequent activity resulting from that spike.

The actual datatype of the implementation is python dictionaries. These data types are mutable objects, and when a neuron passes along its dictionary, it is the reference to the original object that is shared and not a copy. This means that when a neuron, several layers into the PG shares its dictionary to the pre-synaptic neurons own dictionary, there is a trickle-effect that ensures that the entire structure stored in the original input neuron is updated as well.

The root of the entire structure is, as previously mentioned, always an input neuron. It is expected that all input neurons contribute to an input pattern, and therefore, the entire PG initiated by an input consist of the branching structure of all the input neurons. When new input patterns are given to the network, the resulting PGs are compared layer by layer with existing PGs, and if the match percentage for all PGs are below a specified threshold, the new PG is considered unique and is added to the universal list of PGs. If there is a match, the activity is classified as the same as the PG with the highest match percentage.

The specific method for comparing a new PG with any existing PGs first registers the total number of spikes present in the existing PG. This is the baseline for the maximum number of matches between the two. A recursive algorithm then checks each level of each branch in the structure, and counts how many of the neuron IDs that is in the existing PG that are also found in the new PG. Excess spikes in the new PG that are not in the existing one are ignored. The final match is then calculated as $\frac{matches}{totalspikes}$. It is important to reiterate that it is not simply the existence of a spike that constitutes a match, as its place in the structure is crucial.

A visualization of the PG structure can be seen in figure 5.2, along with the corresponding network topology. This PG is the result of a spike in each neuron in the network, with the input neurons strongly connected to the network, and delays chosen so that spikes align at the networks internal post-synaptic neurons. The structure on the left shows the network and its nodes, with colors that corresponds to the neurons place or layer position in the PG. The numbers is the ID of the neuron. The right part of the plot illustrates the PG which encompasses all the activity, starting with the two inputs, and followed by all the successive spikes.

As previously mentioned, one neuron can be in several parts of the PG. In this case, a neuron appears in several places because it is part of the structure of several pre-synaptic neurons. However, since each neuron in this example only fires once, it is actually the same structure corresponding

to the same spike that is seen in multiple places.

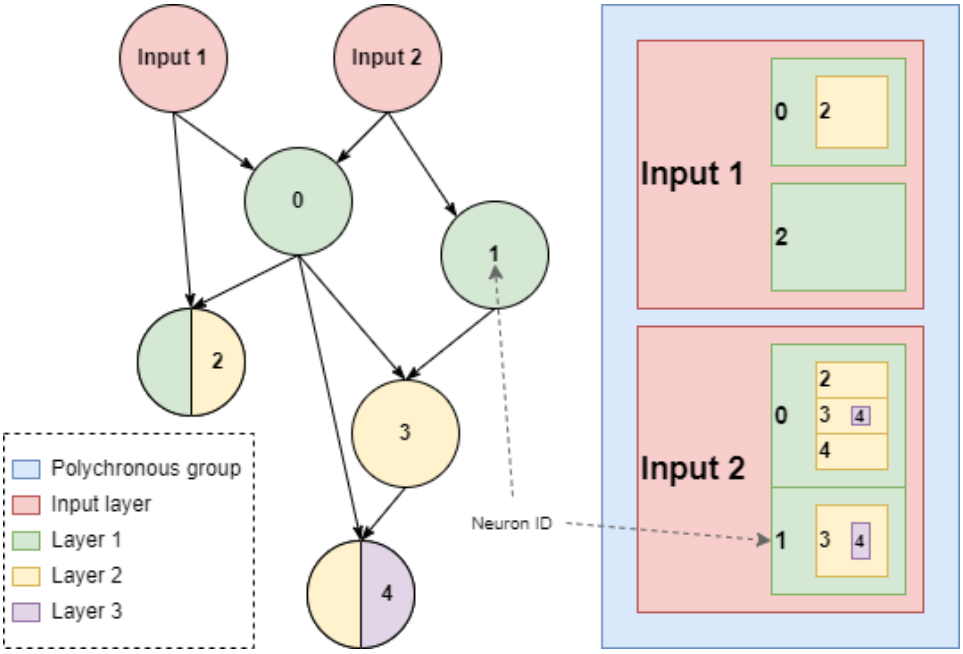

Figure 5.2: Visualization of the datastructure used to store PGs. The color of the nodes, and boxes corresponds to the layer depth in the structure. Some nodes occur in two layers.

Current limitations to the method is that each input can only spike once for each PG, and a limit for the duration of the PG must be set. A new input cannot be given to the network until after this duration and any PG that stretches beyond it is not recorded in the PG. This limits the ways that information can be encoded and given to the network, but the current implementation is intended for inputs that are encoded with the RSL method described in section 2.4.3, which will pose no problems.

The hypothesis is that the learning mechanism is able to adjust for variations of patterns that belong to the same class, while no adjustments to delays will miscategorize these variations.

5.2 Increasing robustness through delay learning

Initially it can be shown that a plastic reservoir can retain activity as an input is increasingly distorted while the activity of an identical but static reservoir seizes. To demonstrate this, a plastic and static reservoir was given a repeating RSL encoded input consisting of four spikes, with an increasing offset between them, while retaining their initial order. This would constitute various instances of a single input class.

Figure 5.3 shows the raster plot of the plastic reservoir on the bottom, and the static reservoir on the top. The delay plasticity allows for the reservoir to gradually adjust the delays to accommodate for the increasing separation between the inputs, retaining reservoir activity past 4000 ms.

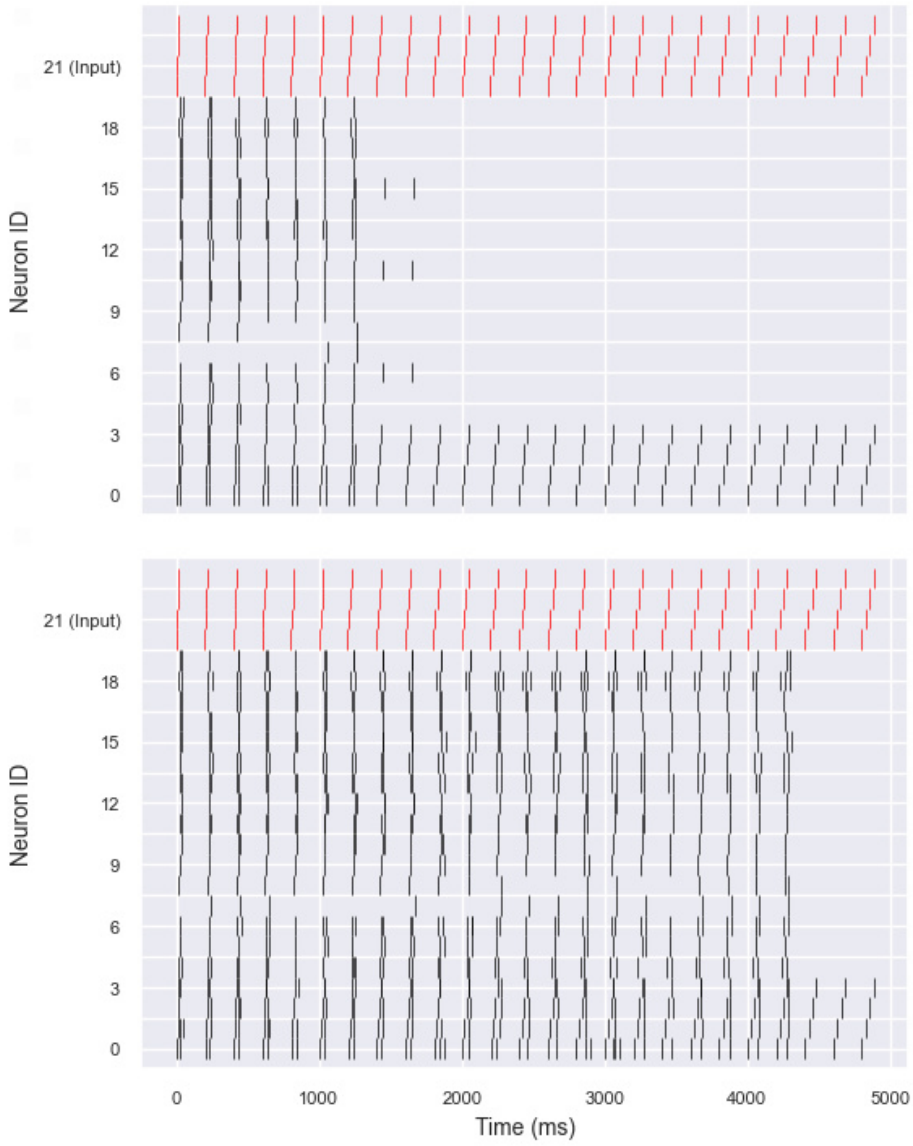

Figure 5.3: The bottom plot shows a plastic reservoir, with adaptive delays that extend the activity. The top plot shows a static reservoir where the activity quickly seizes as the stretched pattern no longer causes reservoir activity. The input spikes are indicated by the red color.

The activity in the static reservoir, however, seizes at around 1000 ms. It is worth noting that the input neurons are strongly connected to one reservoir neuron each, which explains the four spikes that are always present.

This demonstrates the robustness that is introduced by the learning method which sustains activity for a very wide variety of class instances.

5.3 Input classification in feed-forward network

One of the simplest topologies for spiking neural networks is the fully connected feed-forward neural network. In order to make the comparison between the three different topologies more valid, the network size was set to 25 for all topologies, along with 5 input neurons. In the case of the feed-forward topology, the neurons were configured in five layers, with each input neuron connected to one RS neuron in the first layer. The input consisted of 12 instances of one class, followed by a short intermission period before 12 instances of another class was given. An illustration of these classes and instances can be seen in figure 5.1. The reason for having two separate classes as input is to see how the networks handle adjusting for one class, and then following a short intermission period, adjust for a completely new class. The interval between inputs was 400 ms, and the length limit of PGs was set to 200 ms. Allowing for enough time between the termination of the first activity and the onset of the next input pattern is important in order to avoid interference as a result of neuron refractoriness. The threshold for PG matching was set to 0.6.

An example of the network can be seen in figure B.1. The delays were limited to the interval (0,20] ms, and the initialization of the delays were done in the same interval.

Initial testing showed that using a weight of 16 resulted in adequate network activity for this specific configuration. Using lower weights resulted in the activity seizing after the first couple layers.

20 simulations were run with an identical network using internal weights of 16 and various input classes, as previously explained. The results can be seen in table 5.1. The average accuracy over the 20 simulations was 90.4% when delay learning was enabled, while it was 79.4% when delay learning was not enabled.

Table 5.1: Accuracy of identical feed forward network run for 20 simulations with and without delay learning, for various instances of the two input classes.

Network type	Plastic		Static	
Accuracy per input	Class 1	Class 2	Class 1	Class 2
	90.8%	90.0%	76.3%	82.5%
Average accuracy	90.4%		79.4%	

The method for calculating accuracy is illustrated in figure 5.4. The activity resulting from the two classes used for these simulations are indicated by the colored dotted boxes. The PG group that is most prevalent is considered the networks solution to the classification. For the first class it is the green PG, or PG ID 0. The PG method has identified five out of the 12 patterns to belong to this group, indicated by the number in parenthesis. The accuracy for this input class is 41.7%. It is worth noting that the PG ID is just an arbitrary number in order to differentiate between the PG groups. Likewise, for the second input class, nine out of the 12 inputs were

classified as the same, giving an accuracy of 75%.

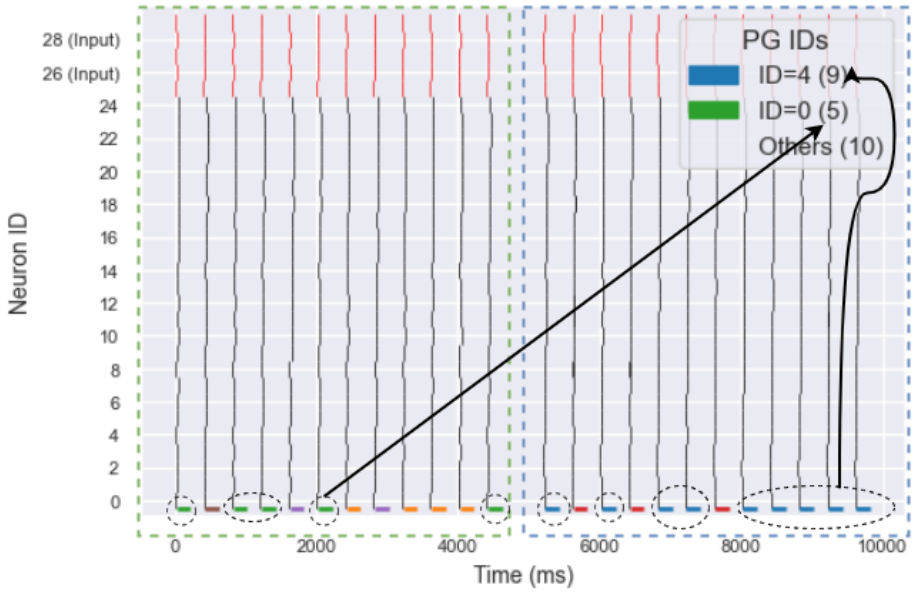

Figure 5.4: The accuracy of the model is calculated by checking how many inputs of one class are classified as the same class.

Figure 5.5 shows the delays as functions of time for a typical simulation, with each color representing the delay of one connection. The delays seem to converge towards the ends of each input sequence, indicated by the red boxes. It is worth noting that the section were the delays seem to have completely converged in the middle of the plot, indicated by the blue boxes, is due to the intermission period between inputs. Likewise, at the end of the plot there is a period where there is no activity. In general the delays are quite evenly distributed with practically no delay saturation and very little diverging behavior.

Although the sample size is low, this is a strong indication that the delay learning method increases classification accuracy in feed-forward neural networks.

5.4 Input classification in ring lattice

Taking another step towards more complex topologies, various unidirectional ring lattice configurations were tested. The relevant parameters that were tested in this case is the connectivity probability between input and network which is limited to the interval [0.1, 0.4], the k feature or out-degree of each neuron in the interval [2,4], and the internal weights from the set 8, 16. Since the ring lattice topology in question is unidirectional, the k feature describes the number of outgoing connections in the anti-clockwise direction. Each configuration also has 3 long-range connections where the outgoing and incoming neuron is randomly chosen. The total number of

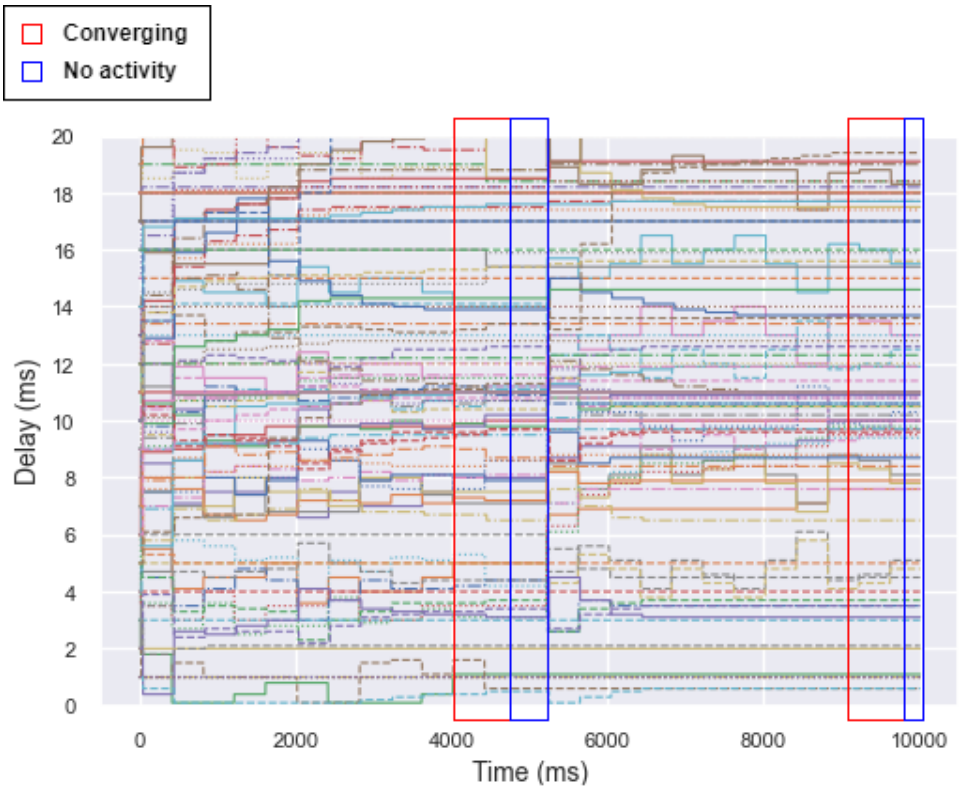

Figure 5.5: Shows delays as functions of time for a typical instance of the 20 simulations. Each color corresponds to the delay of on connection.

neurons is again 25, and there are 5 inputs. The PG matching threshold was set to 0.6. With the described parameter intervals, a total set of 24 configurations were run twice, once with delay learning and once without. The accuracy with delay learning enabled was 86.5%, while it was slightly higher at 90.7% with delay learning disabled.

These numbers do not include the simulations where the network was not able to separate the input which resulted in all instances being classified as the same class. This occurred seven times out of 24 simulations for the plastic network, and six times for the static network. If these instances are counted as zero for both input classes, the accuracy drops down to 61.3% for the plastic network, and 68.1% for the static network.

It is quite likely that increasing the PG matching threshold would reveal that the network is in fact separating the input in these instances, but this would likely lead to lower accuracy in the other simulations. This highlights the potential need for a specific threshold for each network configuration, but it would be quite a laborious task to establish such a threshold for each individual network.

Another potential reason for the separation issue could be low activity, as these instances appear to happen when the input connectivity is quite low. It also appears to occur more frequently for the configurations with lower internal weights. The input separation issue does not seem to

particularly favour either the plastic or static networks. Another interesting observation is that the accuracy appears to be lower for the higher internal weight, which is likely due to the more chaotic behavior as a result of more activity in the network. The entire results table from the simulations can be found in appendix A.1.

All the configurations of ring-lattice networks discussed so far are subject to an identical input. In order to test the effects of various inputs, a promising topology was chosen and another 20 simulations were run with different inputs, in a similar manner as with the feed-forward topology. The configuration that was chosen uses an internal weight of 8, k is set to 3 and the input to reservoir connectivity is 0.3. Figure B.2 shows the ring lattice configuration used for the 20 tests.

Table 5.2: Accuracy of identical ring-lattice network run for 20 simulations with and without delay learning, for various instances of the two input classes.

Network type	Plastic		Static	
Accuracy per input	Class 1	Class 2	Class 1	Class 2
	91.7%	93.3%	97.1%	87.9%
Average accuracy	92.5%		92.5%	

Table 5.2, shows the resulting accuracy of the simulations. The overall accuracy was identical between the static and plastic networks, at 92.5%. There were no class separation issues during these simulations. The lack of difference between the two could be due to the high baseline accuracy of the static network, which does not leave room for a lot of improvement. However, this is a considerable performance increase for the plastic network over the average accuracy seen in appendix A.1. This could be an indication that the learning mechanism is sensitive to the exact ring-lattice configurations.

5.5 Input classification in reservoir

The final and perhaps most complex topology that is tested is the reservoir, which can also be referred to as the recurrent network. This topology is created by iterating through all the neurons of the network and connecting it to every other neuron with a specified probability. The parameters include input to reservoir connectivity in the interval [0.1 , 0.4], reservoir connectivity in the interval [0.1, 0.4], and internal weights from the set 4, 8, 16. Again, the reservoir consist of 25 neurons and there are 5 input neurons.

The accuracy was relatively low for both the static and plastic reservoirs, with an average accuracy of 75.8% and 71.0% respectively. The separation issue was again present, with 11 occurrences out of 44 simulations in both the plastic and static networks. Counting these instances as 0 results in a 53.2% accuracy for the plastic networks and 56.8% for the static networks. Again, the issue was prevalent for networks with low weights and low input and reservoir connectivity, indicating that perhaps low activ-

ity is the culprit. There is also a clear trend of lower accuracy as the weight and connectivity increases, which could result in chaotic behavior leading to low PG matching within the same class.

The entire results table can be found in appendix A.2. The four last configurations using a weight of 16 and reservoir connectivity probability of 0.4 are missing due to memory constraints. This indicates that these configurations resulted in quite chaotic behavior with many large and memory intensive PG structures and few matches.

The reservoir weight of 8 seems to be a good middle ground between the chaotic and low accuracy behavior of the higher weight, and the low activity of the lower weight. A promising configuration with this weight was therefore selected and 20 simulations were run with varying input classes. The specific configuration used a reservoir connectivity of 0.2 and an input connectivity of 0.3.

An example of this configuration can be seen in figure B.3. The overall accuracy was quite high, with the static reservoir achieving an average accuracy of 89.6% while the plastic reservoir achieved a slightly higher accuracy of 91.3%. These results can be seen in table 5.3.

Table 5.3: Accuracy of identical ring-lattice network run for 20 simulations with and without delay learning, for various instances of the two input classes.

Network type	Plastic		Static	
Accuracy per input	Class 1	Class 2	Class 1	Class 2
	89.2%	93.3%	88.8%	90.4%
Average accuracy	91.3%		89.6%	

It is not unexpected that the reservoir in general performs poorly when averaged over several configurations, as this topology is quite sensitive to the choice of parameters. However, it is also clear that the reservoir topology can perform quite well when parameters are carefully chosen.

5.6 Classifying MNIST dataset

This section will showcase a practical application of the delay learning method by comparing the performance of static and plastic networks on classification of images from the MNIST dataset.

The MNIST dataset [8] is commonly used to demonstrate machine learning frameworks and often used as a benchmark for prediction accuracy. The dataset consists of thousands of greyscale 28 by 28 pixel images of handwritten digits.

The RSL encoding method is used to convert the images to spikes so that it can be introduced to the network. Each pixel of the images is converted to a single spike, where the pixel intensity determines the spikes offset from the initiation of the input pattern. By using this method the order and the temporal offset between the spikes describes the differences between the pixels.

Since the images are greyscale with an eight bit color depth, the pixels have integer values in the range [0, 255], where 0 is black and 255 is white. Having offsets in this range is impractical, and therefore the values were scaled down. Various smaller intervals were tested until [0, 40] was found to provide sufficient spacing between input spikes, which allowed the network to separate the input classes.

The method also require that each pixel is represented by one input, and due to computational constraints, it is not feasible to use the original image size. However, reducing the image too much removes a lot of information. A good middle ground avoiding slow computation and too much information loss was to use 10 by 10 pixels. This requires 100 input neurons, and consequently quite large networks.

The method for encoding an image into spikes can be seen in figure 5.6. The downscaled image is seen on the left, which is converted to a matrix in the mentioned range. Row by row, the pixel numbers are then given to the input neurons as spike times. If images are given to the network with a 400 ms interval, then a pixel value of 1 from the first image results in a spike time of 1 ms. The same pixel value for the next image would then give a spike time of 401 ms.

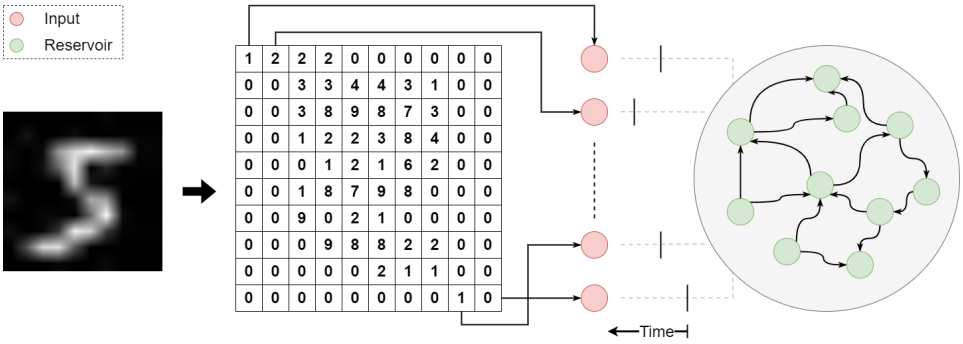

Figure 5.6: The images from the MNIST dataset are initially reduced in size, then the pixel values are scaled down to an appropriate range. Each individual pixel value is then given to one input neuron as a spike time.

Based on the encouraging performance increase shown by the plastic networks over the static networks when using the feed-forward topology in the previous input classification task, this topology was chosen for classifying the MNIST data. A network configuration of 500 RS neurons configured in 5 layers was observed to perform quite well. However, due to the size of the network and the number of inputs, using fully connected layers resulted in chaotic behavior. Through testing, a configuration using weights of 4 and network connectivity of 0.3, yielded good performance. In this case, the network connectivity determines the probability of a connection between two layers.

An interesting result was that in order to achieve good performance with this network, the $G(\Delta t)$ part of the learning mechanism had to be disabled. Figure 5.7 shows the learning method with $G(\Delta t)$ enabled on the top plot, and disabled on the bottom plot. The input for both networks

is the digit 0, repeated 20 times. The top plot shows significantly more activity, and the chaotic behavior results in a large number of different classes. When $G(\Delta t)$ is disabled, however, the intensity of the activity is reduced after a few iterations and the network correctly classifies the remaining inputs as the same class. It is worth noting that other configurations of feed-forward networks did not necessarily require part of the learning method to be disabled, which indicates that the learning method is quite sensitive to the exact network configuration.

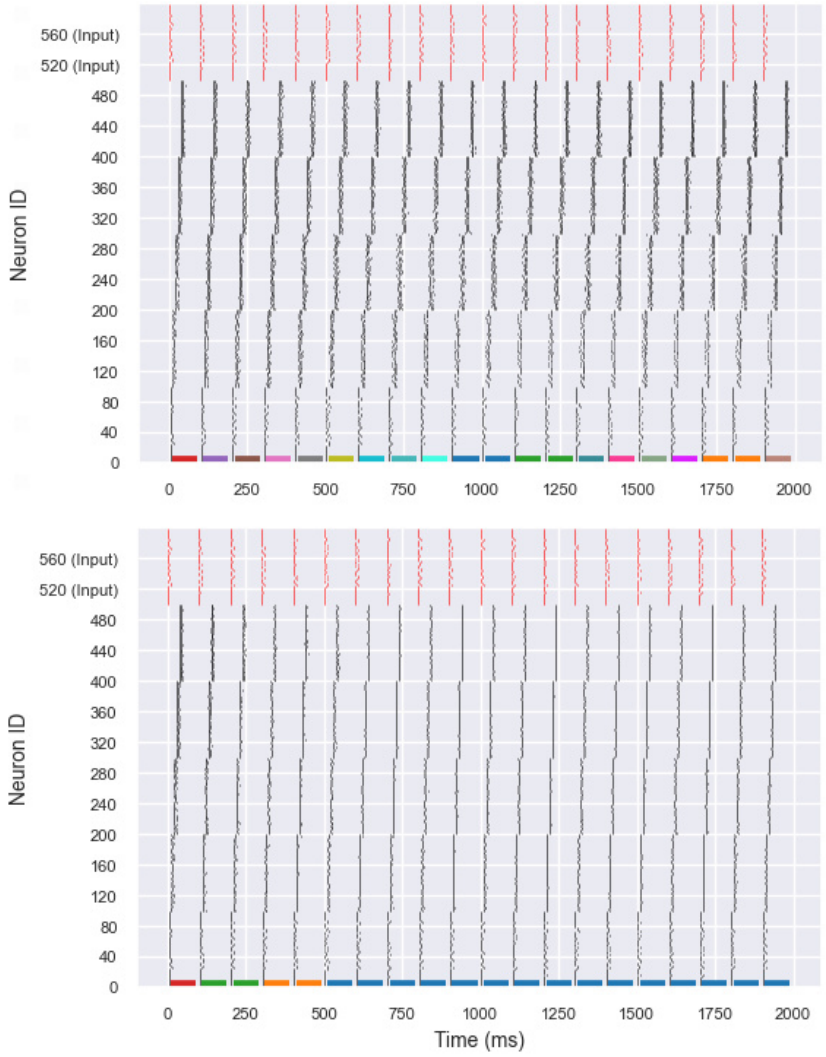

Figure 5.7: The digit five introduced to a network with $G(\Delta t)$ enabled in the top plot, and disabled in the bottom plot.

Adding a second input class and comparing the learning method to its static counterpart, reveals a significant advantage in accuracy in the plastic network over its static counterpart. Figure 5.8 shows the resulting raster

plots, with the static network in the top plot and the plastic network in the bottom plot. The networks are given a series of 20 instances with the digit 0, followed by another series of the digit 7. These specific digits were chosen because they are not very similar in their shape.

The plastic network achieves an accuracy of 75% for the first pattern, and 90% for the second. One interesting observation is that the main PG group of the first input series appears in the beginning of the second input series, before the network adjusts resulting in a new PG group. This means that if the network was static after the first input, it would incorrectly classify all the instances of the second input, substantiating the importance of training with multiple input classes. The static network is quite chaotic and is not able to properly group the activity into meaningful groups.

In order to test the network on an even distribution of the various classes, the same network was tested on each of the 10 digits, with 10 instances of each class. The result can be seen in table 5.4. The plastic network was able to achieve an accuracy of 63.0% while the static network only achieved 28.0%. The raster plots from the simulations can be seen in appendix A.1, which shows than many of the plastic networks appear to settle on a specific class towards the end of the simulation. This means that the accuracy discrepancy between static and plastic networks would likely be further exacerbated if the simulations were run for a longer period.

Table 5.4: Number of correctly classified digits out of 10 instances for each digit for both the static and the plastic network.

Digit	**Static**	**Plastic**
0	4	7
1	3	8
2	2	8
3	2	7
4	1	5
5	6	5
6	2	6
7	2	6
8	3	6
9	3	5
Accuracy	28.0%	63.0%

Overall these results using the feed-forward topology are quite promising. It demonstrates that the learning method is able to consolidate the activity that is the result of similar input classes. However, in order to truly test the method, a large set of randomly chosen and evenly distributes classes should be given to the network in a training phase. Then the delay learning could be disabled and the trained static network would then be tested with unseen data. Due to the relatively time consuming simulations and computationally expensive PG detection method, such a training period is unfeasible with the current implementation and therefore this proof-of-concept will have to suffice. However, implementing a regression

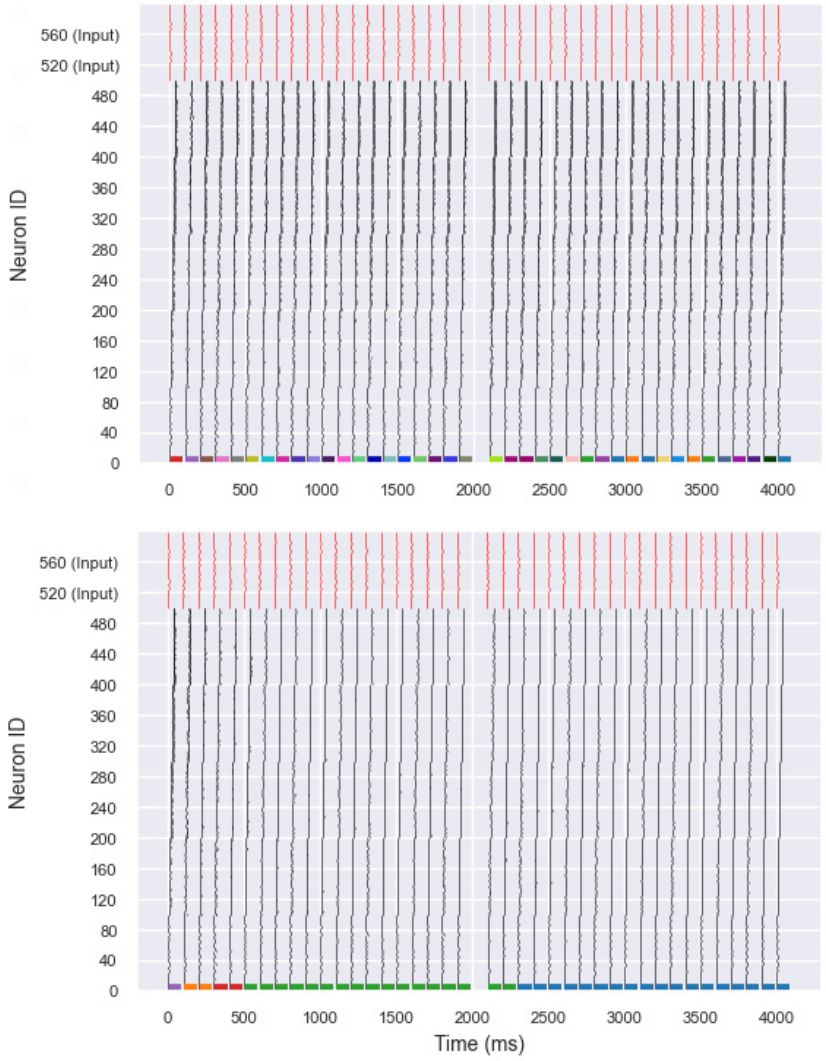

Figure 5.8: Top plot shows a static network, while the bottom plot shows a plastic network. Input consist of 20 instances of the digit 0, followed by 20 instances of the digit 7.

layer for extracting network activity instead of PG detection could make such a training period possible.

Chapter 6

Discussion

The first interesting observation made in this project was the overwhelming trend of diverging delays when rate-coded input was presented to small networks. However, this was not entirely unexpected since the learning method adjusts delays so that each pre-synaptic spike that contributed to the post-synaptic spike would align more towards the average spike arrival time for the next set of similar spikes. The order and offset between these spikes is therefore crucial for what the adjustments to the delays will be. It is simple to imagine how a set of input neurons with different firing rates, will lead to different spike offsets and even spike orders when the spikes arrive at the post-synaptic neuron. This would result in the learning mechanism continuously attempting to adjust the delays to accommodate for the change spike offsets.

Applying RSL encoding, on the other hand, appears to work significantly better. This is a logical result, as this method encodes information in the latencies or offsets between spikes, rather than the rate of spikes. The offset and order of spikes from instances of one class would therefore be somewhat similar, and the delay learning method should be able to adjust to accommodate for these input without drastic changes between instances.

Another interesting observation was that the ring-lattice and reservoir topologies did not show a performance increase when delay learning was applied. This can likely be attributed to the complex temporal nature of networks with recurrent connections. The issue is that a single connection could be active in multiple parts of an activity pattern that is the result of a single input. The average spike arrival time could be different in all these instances, and therefore the delay change would be different, and consequently the delay would never converge.

It was also observed that the learning mechanism could perform well with the ring-lattice and reservoir topologies, when correct parameters were applied. Even a slight improvement in accuracy in the plastic reservoir topology over its static counterpart was observed. This indicate that the proposed learning method is sensitive to the exact configuration of the network.

One improvement to the proposed learning method that could provide a solution to the lack of converging delays is to incorporate a decaying

learning rate. This could be implemented on a per-synapse basis. The proposed implementation is that the learning rate, or change to delays, is reduced for each time the learning mechanism is triggered. If one uses the MNIST dataset as an example, and random images of digits are sequentially introduced to the network, then after some period of time the network will consolidate and become static, and the final delays are the result of gradually decaying incremental changes based on an evenly distributed set of instances from all classes. It is not unlikely that this final static network, regardless of topology, would outperform a static network of randomly chosen delays, since the trained network will hopefully have increased the probability of a certain network pattern to occur as a result of input instances of a specific class.

The proposed learning method introduces changes to the temporal dynamics of the model in two ways, the most obvious being changes to the delays, while the other is by aligning pre-synaptic spikes which increases the magnitude of the input and leads to a quicker post-synaptic response. This means that the post-synaptic spike time is shifted forward in time which could affect the activity the follows this spike in other parts of the network. The proposed solution to this issue, is to align the spikes at a time point, slightly after the average spike arrival time, to compensate for the shift in post-synaptic spike time. This slight shift in alignment would likely depend on the number of contributing spikes, their respective weights and their relative spike arrival times.

Another interesting notion is the inclusion of local weight training through the conventional STDP mechanism. It would be very interesting to observe the network dynamics when local delay learning and weight learning are applied in tandem. Especially exchanging the $G(\Delta t)$ part of the delay learning mechanism with synaptic depression could yield interesting results. Then the effect of the spike would be decreased instead of being shifted backwards in time.

It was observed that the learning method in its current state is quite sensitive to the exact configuration of the network, especially for ring-lattice and reservoir networks. Applying a genetic algorithm to find an optimal learning method that could potentially perform well on a wider range of network configurations and topologies would be an interesting endeavor. The parameter space could include parameters related to the current learning method from equations 4.1 and 4.3, disabling $G(\Delta t)$, learning rates, decaying learning rates, inclusion of STDP or other local delay learning methods entirely.

When working with AI, one should always consider the ethical implications. As for any implementation of AI, the goal is to achieve a model or method that can be applied in real-life scenarios. This can raise issues related to the explainability of the model, as many are reluctant to using black-box models. This is a difficult issue to overcome as the spiking networks can be quite large and the activity complicated.

A positive ethical result of this work is that SNN's are more power-efficient, as previously stated, which can reduce impacts on the environment. When the neurons and synapses are implemented as discrete elec-

trical components the efficiency over conventional computers is drastically increased as the components only expend energy when they are active. This is in contrast to the inefficient synchronous clock used by classical computers which forces all computational units to respond even it they are not part of the computation.

Chapter 7

Conclusion

The purpose of this project was to develop a local delay learning method that adjusts delays based on local knowledge, in order to improve the performance of spiking neural networks. A delay learning method was developed, and subsequently proved to increase performance of feed-forward networks on classification tasks.

The learning method consist of two mechanisms. The first mechanism adjust delays of connections that are contributing pre-synaptic spikes that are causally related to a post-synaptic spike to improve spike alignment. The theory is that by improving the alignment of these spikes, the chance of eliciting the same post-synaptic response from similar inputs is increased. This would help the network separate different classes of inputs. The second mechanism pushed away spikes that are not causally related to the post-synaptic spike by increasing the connection delay.

Two methods for input encoding were tested on simple networks with the learning method applied. The results showed that rate-coded inputs leads to diverging delays and is not a suitable encoding method for the proposed learning mechanism.

The RSL encoding method showed significantly better response in delay behavior. This method was subsequently used to encode inputs for three topologies, in order to compare the performance of the learning method on various network configurations.

The network configurations were tested on a classification task, where a novel PG detection method was developed and used to group network behavior into classes. The three topologies used was the feed-forward, ring-lattice and reservoir. Various versions of these topologies was tested with and without delay learning enabled. The only topology that showed a substantial performance increase when delay learning was applied, was the feed-forward networks.

The feed-forward topology was then used to classify hand-written digits from the MNIST dataset, and the plastic networks showed a significant performance increase in this classification task over the static networks.

The project highlights the difficulty of developing delay learning methods for recurrent networks. This is an obstacle that is important

to overcome, as recurrent SNNs are powerful computational frameworks that can perform well on sequential data tasks due to their capacity for memory. It is also clear that the method used for encoding inputs is an important consideration for temporal learning methods. Likewise, the way the network output is interpreted is likely just as important, although only one method for decoding output was tested in this project.

The work presents a small but important step in attempting to fill a void in current research pertaining to local learning of delays in spiking neural networks. Although the results of the project indicates promising performance on specific topologies, much work remains to develop an optimal local learning mechanism for delays that can be applied to more complicated tasks and on more interesting topologies.

Bibliography

[1] Andrew Allot and Midorff David. *IB biology. Course book*. Oxford : Oxford University Press, 2014., 2015.

[2] Guillaume Bellec et al. 'Long short-term memory and learning-to-learn in networks of spiking neurons'. In: (2018).

[3] S. Bohté, J. Kok and H. L. Poutré. 'SpikeProp: backpropagation for networks of spiking neurons'. In: *ESANN*. 2000.

[4] Dirk Bucher and Jean-Marc Goaillard. 'Beyond faithful conduction: short-term dynamics, neuromodulation, and long-term regulation of spike propagation in the axon'. eng. In: *Progress in neurobiology* 94.4 (2011), pp. 307–346. ISSN: 0301-0082.

[5] CE Carr and M Konishi. 'A circuit for detection of interaural time differences in the brain stem of the barn owl'. eng. In: *The Journal of neuroscience* 10.10 (1990), pp. 3227–3246. ISSN: 0270-6474.

[6] Joseph Chrol-Cannon, Yaochu Jin and André Grüning. 'An efficient method for online detection of polychronous patterns in spiking neural networks'. In: *Neurocomputing* 267 (Dec. 2017), pp. 644–650. DOI: 10.1016/j.neucom.2017.06.025. URL: https://doi.org/10.1016%5C%2Fj.neucom.2017.06.025.

[7] Dominique Debanne. 'Information processing in the axon'. In: *Nat Rev Neurosci* 5.4 (2004), pp. 304–316. ISSN: 1471-003X. DOI: 10.1038/nrn1397.

[8] Li Deng. 'The mnist database of handwritten digit images for machine learning research'. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[9] Roger M Enoka and Jacques Duchateau. 'Rate Coding and the Control of Muscle Force'. eng. In: *Cold Spring Harbor perspectives in medicine* 7.10 (2017), a029702. ISSN: 2157-1422.

[10] Richard FitzHugh. 'Impulses and Physiological States in Theoretical Models of Nerve Membrane'. eng. In: *Biophysical journal* 1.6 (1961), pp. 445–466. ISSN: 0006-3495.

[11] W Gerstner et al. *A neuronal learning rule for sub-millisecond temporal coding*. eng.

[12] Tim GOLLISCH and Markus MEISTER. 'Rapid Neural Coding in the Retina with Relative Spike Latencies'. eng. In: *Science (American Association for the Advancement of Science)* 319.5866 (2008), pp. 1108–1111. ISSN: 0036-8075.

[13] Y Grossman, I Parnas and M E Spira. 'Differential conduction block in branches of a bifurcating axon'. eng. In: *The Journal of physiology* 295.1 (1979), pp. 283–305. ISSN: 0022-3751.

[14] H Hatt and D O Smith. 'Synaptic depression related to presynaptic axon conduction block'. eng. In: *The Journal of physiology* 259.2 (1976), pp. 367–393. ISSN: 0022-3751.

[15] Simon Haykin. *Neural networks and learning machines*. eng. Upper Saddle River, N.J, 2009.

[16] A. L. Hodgkin and A. F. Huxley. 'A quantitative description of membrane current and its application to conduction and excitation in nerve'. In: *Bulletin of mathematical biology* 52.1 (1990), pp. 25–71. ISSN: 0092-8240. DOI: 10.1016/S0092-8240(05)80004-7.

[17] Kurt Hornik. 'Approximation capabilities of multilayer feedforward networks'. eng. In: *Neural networks* 4.2 (1991), pp. 251–257. ISSN: 0893-6080.

[18] E. M. Izhikevich. 'Simple model of spiking neurons'. In: *IEEE Trans Neural Netw* 14.6 (2003), pp. 1569–1572. ISSN: 1045-9227. DOI: 10.1109/TNN.2003.820440.

[19] E. M. Izhikevich. 'Which model to use for cortical spiking neurons?' In: *IEEE Trans Neural Netw* 15.5 (2004), pp. 1063–1070. ISSN: 1045-9227. DOI: 10.1109/TNN.2004.832719.

[20] Eugene M Izhikevich. 'Resonate-and-fire neurons'. eng. In: *Neural networks* 14.6 (2001), pp. 883–894. ISSN: 0893-6080.

[21] Eugene M. Izhikevich. 'Polychronization: Computation with Spikes'. In: *Neural Comput* 18.2 (2006), pp. 245–282. ISSN: 1530-888X,0899-7667. DOI: 10.1162/089976606775093882.

[22] Eugene M. Izhikevich. *Simple model of spiking neurons code example 1*. https://www.izhikevich.org/publications/net.m. 2003.

[23] Eugene M. Izhikevich. *Simple model of spiking neurons code example 2*. https://www.izhikevich.org/publications/figure1.m. 2003.

[24] S.P Johnston et al. 'A Hybrid Learning Algorithm Fusing STDP with GA based Explicit Delay Learning for Spiking Neurons'. eng. In: *2006 3rd International IEEE Conference Intelligent Systems*. IEEE, 2006, pp. 632–637. ISBN: 9781424401956.

[25] Jaehyun Kim et al. 'Deep neural networks with weighted spikes'. eng. In: *Neurocomputing (Amsterdam)* 311 (2018), pp. 373–386. ISSN: 0925-2312.

[26] P. E. Latham et al. 'Intrinsic Dynamics in Neuronal Networks. I. Theory'. In: *Journal of Neurophysiology* 83.2 (2000). PMID: 10669496, pp. 808–827. DOI: 10.1152/jn.2000.83.2.808. eprint: https://doi.org/10.1152/jn.2000.83.2.808. URL: https://doi.org/10.1152/jn.2000.83.2.808.

[27] Dingbang Liu, Hao Yu and Yang Chai. 'Low-Power Computing with Neuromorphic Engineering'. eng. In: *Advanced intelligent systems* 3.2 (2021), 2000150–n/a. ISSN: 2640-4567.

[28] C. Luscher et al. 'Action potential propagation through embryonic dorsal root ganglion cells in culture. I. Influence of the cell morphology on propagation properties'. In: *Journal of Neurophysiology* 72.2 (1994). PMID: 7983524, pp. 622–633. DOI: 10.1152/jn.1994.72.2.622. eprint: https://doi.org/10.1152/jn.1994.72.2.622. URL: https://doi.org/10.1152/jn.1994.72.2.622.

[29] Wolfgang Maass. 'Networks of spiking neurons: The third generation of neural network models'. In: *Neural networks* 10.9 (1997), pp. 1659–1671. ISSN: 0893-6080. DOI: 10.1016/S0893-6080(97)00011-7.

[30] Warren S. McCulloch and Walter Pitts. 'A logical calculus of the ideas immanent in nervous activity'. In: *The Bulletin of Mathematical Biophysics* 5.4 (1943), pp. 115–133. ISSN: 0007-4985. DOI: 10.1007/BF02478259.

[31] C Morris and H Lecar. 'Voltage oscillations in the barnacle giant muscle fiber'. eng. In: *Biophysical journal* 35.1 (1981), pp. 193–213. ISSN: 0006-3495.

[32] Flavio öhlich. *Network Neuroscience*. eng. San Diego: Elsevier Science Technology, 2016. ISBN: 9780128015605.

[33] Seongsik Park et al. 'T2FSNN: Deep Spiking Neural Networks with Time-to-first-spike Coding'. In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 2020, pp. 1–6. DOI: 10.1109/DAC18072.2020.9218689.

[34] Hélène Paugam-Moisy, Régis Martinez and Samy Bengio. 'Delay learning and polychronization for reservoir computing'. eng. In: *Neurocomputing (Amsterdam)* 71.7 (2008), pp. 1143–1158. ISSN: 0925-2312.

[35] D. I. Perrett, E. T. Rolls and W. Caan. 'Visual neurones responsive to faces in the monkey temporal cortex'. In: *Exp Brain Res* 47.3 (1982), pp. 329–342. ISSN: 0014-4819. DOI: 10.1007/BF00239352.

[36] Filip PONULAK and Andrzej KASINSKI. 'Supervised Learning in Spiking Neural Networks with ReSuMe: Sequence Learning, Classification, and Spike Shifting'. eng. In: *Neural computation* 22.2 (2010), pp. 467–510. ISSN: 0899-7667.

[37] Dale Purves and Stephen Mark. Williams. *Neuroscience. 2nd edition*. Sinauer Associates, 2001.

[38] F Rosenblatt. 'The perceptron: A probabilistic model for information storage and organization in the brain'. eng. In: *Psychological review* 65.6 (1958), pp. 386–408. ISSN: 0033-295X.

[39] B. Schrauwen and J. Van Campenhout. 'Extending SpikeProp'. In: *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*. Vol. 1. 2004, pp. 471–475. DOI: 10.1109/IJCNN.2004.1379954.

[40] Benjamin Schrauwen and Jan Van Campenhout. 'Extending SpikeProp'. eng. In: *IEEE International Joint Conference on Neural Networks (IJCNN)*. Budapest, Hungary: IEEE, 2004, pp. 471–475. ISBN: 0-7803-8359-1.

[41] Gregory D. Smith et al. 'Fourier Analysis of Sinusoidally Driven Thalamocortical Relay Neurons and a Minimal Integrate-and-Fire-or-Burst Model'. eng. In: *Journal of Neurophysiology* 83.1 (2000), pp. 588–610. ISSN: 0022-3077.

[42] Sen Song, Kenneth Miller and L.F. Abbott. 'Competitive Hebbian learning through spike timing-dependent plasticity'. In: *Nature neuroscience* 3 (Oct. 2000), pp. 919–26. DOI: 10.1038/78829.

[43] David Sterratt et al. *Principles of Computational Modelling in Neuroscience*. eng. Cambridge: Cambridge University Press, 2011. ISBN: 9780521877954.

[44] Aboozar Taherkhani et al. 'A Supervised Learning Algorithm for Learning Precise Timing of Multiple Spikes in Multilayer Spiking Neural Networks'. eng. In: *IEEE transaction on neural networks and learning systems* 29.11 (2018), pp. 5394–5407. ISSN: 2162-237X.

[45] Aboozar Taherkhani et al. 'DL-ReSuMe: A Delay Learning-Based Remote Supervised Method for Spiking Neurons'. In: *IEEE Trans Neural Netw Learn Syst* 26.12 (2015), pp. 3137–3149. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2015.2404938.

[46] Jacques Thorpe Simon; Gautrais. 'Rank order coding'. In: *Computational Neuroscience: Trends in Research* (1998), pp. 113–118. ISSN: 978-1-4613-7190-8. DOI: 10.1007/978-1-4615-4831-7_19.

[47] Xiangwen Wang, Xianghong Lin and Xiaochao Dang. 'A Delay Learning Algorithm Based on Spike Train Kernels for Spiking Neurons'. eng. In: *Frontiers in neuroscience* 13 (2019), pp. 252–252. ISSN: 1662-4548.

[48] Xiangwen Wang, Xianghong Lin and Xiaochao Dang. 'Supervised learning in spiking neural networks: A review of algorithms and evaluations'. In: *Neural Netw* 125 (2020), pp. 258–280. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2020.02.011.

[49] James C. R. Whittington and Rafal Bogacz. 'Theories of Error Back-Propagation in the Brain'. In: *Trends Cogn Sci* 23.3 (2019), pp. 235–250. ISSN: 1364-6613. DOI: 10.1016/j.tics.2018.12.005.

[50] Hugh R Wilson. 'Simplified Dynamics of Human and Mammalian Neocortical Neurons'. eng. In: *Journal of theoretical biology* 200.4 (1999), pp. 375–388. ISSN: 0022-5193.

[51] Fleur Zeldenrust, Wytse J. Wadman and Bernhard Englitz. 'Neural Coding With Bursts—Current State and Future Perspectives'. In: *Frontiers in Computational Neuroscience* 12 (2018). ISSN: 1662-5188. DOI: 10.3389/fncom.2018.00048. URL: https://www.frontiersin.org/article/10.3389/fncom.2018.00048.

[52] Han Zhang et al. 'StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks'. eng. In: (2016).

Appendix A

Input classification

A.1 Ring lattice

Ring lattice						
resW	k	inpP	Plastic		Static	
			Pattern 1	Pattern 2	Pattern 1	Pattern 2
8	2	0.1	-	-	-	-
8	2	0.2	-	-	-	-
8	2	0.3	12	12	12	12
8	2	0.4	12	12	12	12
8	3	0.1	-	-	-	-
8	3	0.2	-	-	-	-
8	3	0.3	12	12	12	12
8	3	0.4	11	8	9	11
8	4	0.1	-	-	-	-
8	4	0.2	12	12	12	12
8	4	0.3	11	12	9	12
8	4	0.4	8	12	6	11
16	2	0.1	12	12	12	12
16	2	0.2	-	-	-	-
16	2	0.3	12	6	12	12
16	2	0.4	12	12	12	12
16	3	0.1	8	12	7	12
16	3	0.2	-	-	12	11
16	3	0.3	12	12	12	12
16	3	0.4	12	12	12	12
16	4	0.1	6	3	12	12
16	4	0.2	11	12	8	12
16	4	0.3	6	11	11	12
16	4	0.4	5	7	7	5
Average accuracy			86.5%		90.7%	
Average accuracy w/null			61.3%		68.1%	

A.2 Reservoir

Reservoir						
resW	resP	inpP	Plastic		Static	
			Pattern 1	Pattern 2	Pattern 1	Pattern 2
4	0.1	0.1	-	-	-	-
4	0.1	0.2	-	-	-	-
4	0.1	0.3	-	-	-	-
4	0.1	0.4	12	12	12	12
4	0.2	0.1	-	-	-	-
4	0.2	0.2	-	-	-	-
4	0.2	0.3	12	12	12	12
4	0.2	0.4	12	12	12	6
4	0.3	0.1	-	-	-	-
4	0.3	0.2	12	12	12	12
4	0.3	0.3	8	12	9	12
4	0.3	0.4	6	8	6	8
4	0.4	0.1	-	-	-	-
4	0.4	0.2	12	8	12	12
4	0.4	0.3	8	9	7	8
4	0.4	0.4	6	12	12	9
8	0.1	0.1	-	-	-	-
8	0.1	0.2	-	-	-	-
8	0.1	0.3	12	5	12	12
8	0.1	0.4	6	12	12	6
8	0.2	0.1	12	12	12	12
8	0.2	0.2	12	11	12	12
8	0.2	0.3	9	12	12	9
8	0.2	0.4	7	8	7	9
8	0.3	0.1	12	12	12	12
8	0.3	0.2	12	12	7	11
8	0.3	0.3	6	6	7	9
8	0.3	0.4	6	9	9	5
8	0.4	0.1	12	9	12	8
8	0.4	0.2	9	5	5	9
8	0.4	0.3	9	6	8	9
8	0.4	0.4	3	9	6	6
16	0.1	0.1	-	-	-	-
16	0.1	0.2	-	-	-	-
16	0.1	0.3	12	3	12	12
16	0.1	0.4	12	9	12	8
16	0.2	0.1	12	12	12	12
16	0.2	0.2	8	5	6	12
16	0.2	0.3	6	9	4	7
16	0.2	0.4	7	5	10	5
16	0.3	0.1	3	5	6	4
16	0.3	0.2	3	2	6	10
16	0.3	0.3	2	3	3	2
16	0.3	0.4	2	2	8	2
Average accuracy			71.0%		75.8%	
Average accuracy w/null			53.2%		56.8%	

A.3 MNIST classification

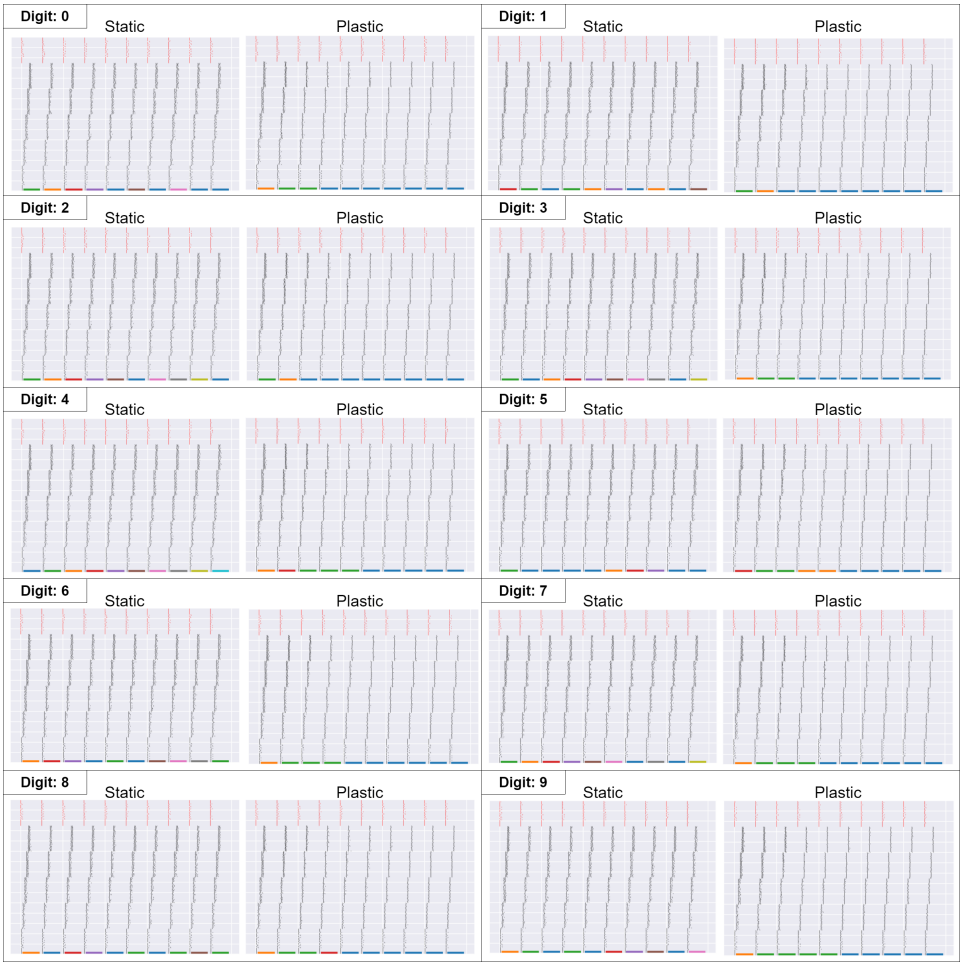

Figure A.1: Classification of MNIST digits using feed-forward topology. Similar colors indates that the spike activity belongs to the same PG.

Appendix B

Topologies

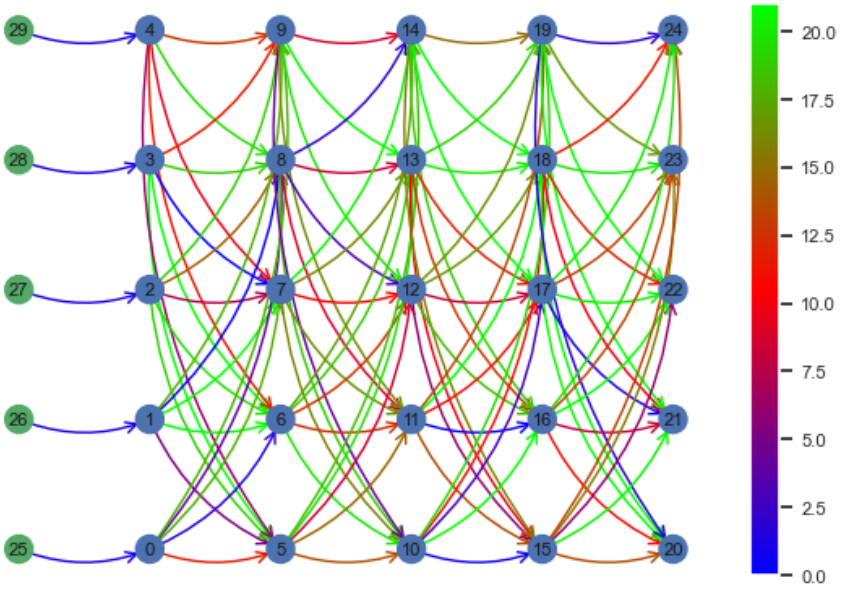

Figure B.1: An example of the feed-forward topology with green nodes representing inputs and blue nodes representing RS neurons. Edge colors indicate delays in ms.

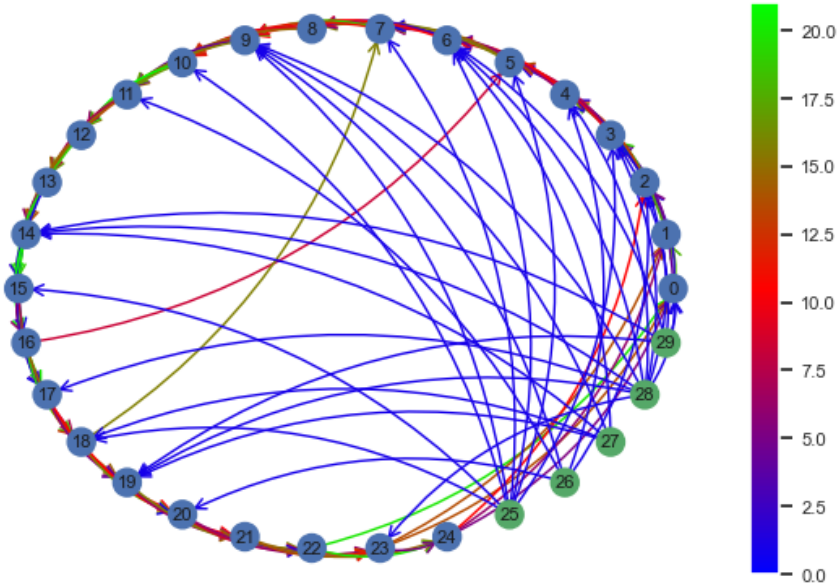

Figure B.2: A typical ring lattice topology.

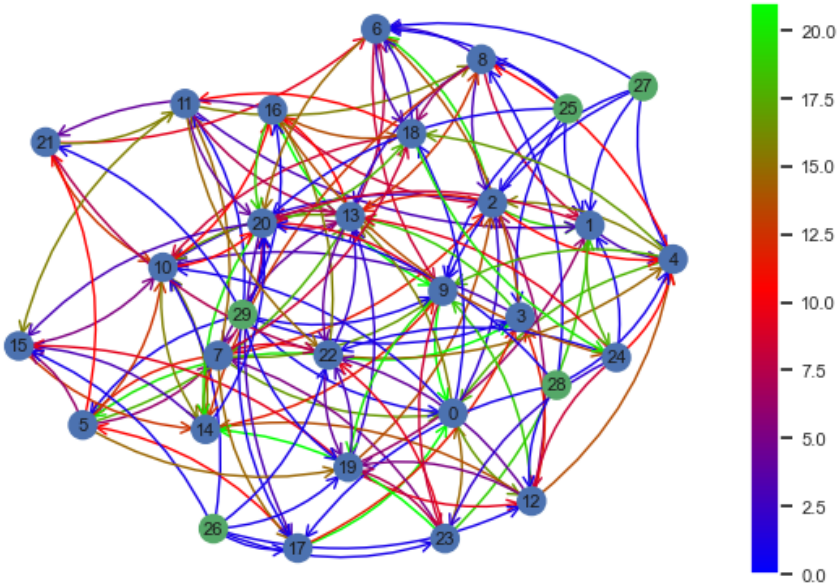

Figure B.3: A typical reservoir topology.