TOWARDS MODELLING OF AUTONOMOUS NEUROMORPHIC LEARNING SYSTEMS.

A THESIS SUBMITTED TO

THE UNIVERSITY OF KENT

IN THE SUBJECT OF COMPUTER SCIENCE

FOR THE DEGREE

OF PHD.

By

Jakub Fil

October 27, 2021

Abstract

This thesis aims to investigate physically plausible models of spiking neurons and propose a path for autonomous molecular implementation.

First, I will discuss supervised learning of multi-class stimuli in a single spiking neuron. Particularly, I will focus on the aggregate-label learning framework originally proposed by Gütig (2016). To this end, I will introduce a novel model of a spiking neuron capable of performing complex computational tasks, while remaining simple and readily interpretable. Moreover, I will demonstrate how this neuronal model can be interpreted as a chemical reaction network, and how synaptic weights can be encoded by reaction rate constants.

Next, I will investigate a minimal molecular model of a spiking neuron capable of unsupervised learning. In order to be practically useful, such molecular implementation needs to be autonomous. I will define what it means for the learning systems to be autonomous, and propose a model which implements both the neuronal functions as well as the learning algorithm within a chemical reaction network. Through extensive simulations I will demonstrate that this model is capable of autonomous recognition of frequency biases and temporal correlations embedded into discrete spike trains.

Lastly, I will present an implementation of a spiking neuron based on DNA-strand displacement interactions. The advantage of this method is that it can realistically be synthesised in a laboratory. The DNA neuron will be shown to be capable of performing a variety of computational tasks including temporal correlation learning and novelty detection.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The nervous systems consist of interconnected networks of billions of cells - neurons. There are many different types of neurons in the brain, however their basic function remains to integrate input signals and propagate outputs forward into the network. Neurons can only relay information one way, they accumulate electrical pulses through their dendrites and under certain conditions propagate the signal through their axon to neighbouring cells. When a neuron activates in correlation with its neighbour, their synaptic connection is strengthened. In simple terms, their synaptic plasticity follows a well-known Hebbian principle - "what fires together, wires together" (Hebb 1950). These, in essence very simple, devices when combined in networks are capable of producing intelligent behaviour.

The study of artificial neural networks (ANNs) and an effort to understand the process of learning and memory formation have a long history. Apart from the establishment of the Hebbian principle by Hebb (1950), perhaps the first leap in our understanding of these systems was the introduction of the perceptron model (Rosenblatt 1958), which mathematically formulated how information storage and network interactions could be implemented in the brain. This concept led to an enormous research effort in the following years, especially after the discovery of the backpropagation algorithm, initially described by Linnainmaa (1976), which allowed for training of deep

multi-layered networks (Rumelhart, Hinton and Williams 1986). Nowadays, the ANNs became widely adopted in variety of use-cases and industries. Networks with billions or even trillions of weights are becoming increasingly common, for example in natural language processing (Fedus, Zoph and Shazeer 2021). Nevertheless, using the models of this size leads to extremely high energy costs when training them, and becomes increasingly difficult to validate due to their hardware requirements (Brown et al. 2020).

Spiking neural networks (SNNs) are a biologically plausible alternative to the ANNs. Unlike standard artificial neuronal networks which are rate-coded, in the SNNs the information is conveyed in the form of discrete spike signals. This type of information processing has been shown to be advantageous over traditional approaches, for example due to their significantly shorter time-scale of processing input features (Johansson and Birznieks 2004). Moreover, the SNNs can be trained in an unsupervised way by employing biologically inspired learning rules such as spike-time dependent plasticity (STDP). The state of a spiking neuron is defined by spatial, as well as temporal sequences of pre-synaptic neuronal activations (Abbott, DePasquale and Memmesheimer 2016). This allows to encode the spike-based signals via temporal coding. In contrast to the rate-coded approaches, this framework enables a neuron to distinguish signals based on their temporal correlations, hence allowing for a more refined signal filtering. Arguably, this is not only more realistic with respect to the real neuronal systems (Brette 2015), but also a single spiking neuron was conjectured to be computationally more powerful than a single rate-coded neuron. Notably, Maass (1997) claimed that "a single spiking neuron is able to replace hundreds of hidden units on a sigmoidal neural network". One example of such increased capabilities could be learning multiple classes of patterns by a single spiking neuron and recognising them by means of releasing a different amount of output spikes (Gütig 2016). These features suggest that the SNNs may offer a greater potential for fast and efficient AI.

Multiple different models of spiking neurons have been proposed over the years. Notably, some of them focused on being true to nature, while others were designed to

fit the requirements of neuromorphic computing platforms. The examples include simple and widely applicable models like leaky integrate-and-fire (Brunel and van Rossum 2007), models which allow for derivation of sophisticated learning rules - tempotron (Gütig and Sompolinsky 2006; Gütig 2016), as well as models which closely replicate realistic dynamic behaviour of biological neurons and capture well recognised firing patterns seen in the brain - Hodgkin-Huxley neuron (Hodgkin and Huxley 1952), Izhikevich neuron (Izhikevich 2003), and spike-response model (Gerstner and Kistler 2002b).

One of the aspects which are often considered while designing a spiking neuron model is its biological plausibility. While appealing, the precise definition of what does it mean to be biologically plausible is often vague. There are many different aspects which may be considered essential for such a neuron to be functional. These may include neuronal functions such as refractory period, post-spike membrane reset, or discrete spiking. Nevertheless, not all models encompass all of those elements. For example, models like the multi-spike tempotron (MST) (Gütig 2016) offer a powerful framework for learning multi-class stimuli, however they are also internally complex. This vastly limits its potential, by making the model difficult to train and deploy outside of the simulation on a digital computer.

Spiking neurons are the most useful when simulated on specialised *neuromorphic* hardware. These computing platforms are designed to efficiently simulate large populations of neurons with biologically plausible properties. Due to the discrete nature of spiking neurons, the electrical signals in the processor only need to be propagated at the time of firing. This brings an advantage over traditional approaches, in that it significantly reduces the energy consumption of both training and simulation of these neural networks. In some contexts this approach could also improve the speed of computation, and therefore allows for real-time execution of large populations of realistic neural networks (Furber and Bogdan 2020), which is especially important for modelling the dynamics of human brain. Moreover, this feature is also crucial for systems which need

to interact with the outside environment, such as robots or self-driving cars.

While applications which interact with the outside environment generate a lot of interest, there is also a wealth of possible use-cases which require computational systems which need to perform on the molecular scale. Such computational devices are of interest in a number of practical tasks, for example in targeted drug delivery (Ausländer, Wieland and Fussenegger 2012). This perspective encourages the development of learning systems which would be innate to the biochemical environment within the living organisms. Additionally, such system would need to be physically realisable and be able to independently interact with its surroundings in order to detect environmental cues.

Even though intelligence and learning are often only associated with organisms with neuronal circuits some organisms, such as bacteria, have the ability to analyse and adapt to changes in their environment. In this case, these intelligent patterns of behaviour are implemented through bio-molecular circuits in their entirety. The field of synthetic biology has uncovered some of the mechanistic properties of these organisms, and proposed frameworks for implementation of artificial cells which can exhibit complex behaviours, for example associative learning (Fernando et al. 2009) or classification (Blount et al. 2017). Recent advancements in bio-engineering bring forward the possibility to synthesise intelligent artificial cells which could perform many useful functions by interacting with the biochemical environment, for example within the human body. Nevertheless, designing such systems has proven to be a challenge due to a range of requirements, such as the need for the system to be low-powered, small, and autonomous.

Autonomy may be one of the most important aspects which need to be considered while designing a learning system in *wet-ware*. When these systems are embedded into a living organism or synthesised in a wet-lab they need to be able to operate without any influence from the external observer. This poses a number of challenges. Most importantly, this type of system needs to learn in an unsupervised fashion. Moreover,

these systems would not have any digital memory to store information, hence they cannot rely on algorithms which require the knowledge of their input history. The learning system also needs to encompass mechanisms such as signal modulation, i.e. scaling of the pre-synaptic input proportional to their weights. Importantly, the lack of digital memory would also mean that these weights need to be represented internally to the system, rather than stored outside. Notably, the current interpretations of spiking neurons typically only model signal integration of temporal inputs and threshold-mediated output firing. It is assumed that an external observer applies a learning algorithm, such as STDP, when a neuron fires, and readjust the synaptic weights based on the history of pre-synaptic inputs. Therefore, such a learning framework would be difficult to realise in synthetic biology.

Before attempting to build such synthetic bio-chemical computers, it is necessary to first understand how to program them. A common approach is to use gene regulatory networks (GRNs) (Racovita and Jaramillo 2020). Fernando et al. (2009) used GRNs to implement an artificial single-celled organism capable of associative learning, however their model was limited to a predefined coincidence pattern and could not be extended to accommodate more advanced computation. Programming GRNs still poses a number of difficulties, one being non-specific binding between biochemical molecules and resulting cross-talk which makes them difficult to control (Grah and Friedlander 2020; Grant et al. 2016). Perhaps a more promising biological programming language could be found in DNA. In particular, the framework of DNA-strand displacement (DSD) has proven to be a reliable substrate for computation. DSD systems have been shown to be capable of universal computation (Seelig et al. 2006), and in principle can implement any set of chemical interactions (Chen et al. 2013). More importantly, nowadays arbitrary DNA strands can be efficiently synthesised in a laboratory and their dynamic behaviour can be accurately predicted from simulations (Yurke et al. 2000; Fontana 2006).

1.1 Thesis outline

The central research question of this thesis is to uncover the necessary components for computation in spiking neurons and present a possible approach to modelling neuromorphic systems in the context of synthetic biology. It will also address the question of autonomy which is revealed to be crucial for such implementations. Moreover, I will discuss what constitutes a biologically plausible neuromorphic model, and propose a path towards experimental *in-vivo* implementation through DNA-based computing.

To this end, I will first review the most prominent spiking neuron models, training algorithms, and discuss the advantages of using neuromorphic platforms. The rest of the background will cover computation within single-celled organisms, both biological and artificial, as well as introduce the paradigm of DNA-based computing, in particular the DNA-strand displacement framework.

In the first research chapter of the thesis, I will focus on supervised learning of multi-class stimuli in a single spiking neuron. In particular, I will examine the aggregate-label learning framework for multi-spike tempotron (Gütig 2016). The main contribution of this chapter is a neuron model which can perform the same tasks as the MST - the generalised neuronal model (GNM). The GNM's performance will be compared to other neuron models trained to recognise multiple classes of input patterns. An alternative, less biologically plausible but more efficient, training algorithm will be discussed as well. I will also demonstrate how such neurons could be arranged in networks and trained using an algorithm similar to error backpropagation. Lastly, using the fact that the GNM can be described as a system of differential equations, I will demonstrate how this neuron model can be interpreted as a chemical reaction network, and how synaptic weights can be encoded by its reaction rate constants.

In the following chapter, I will propose a fully autonomous neuronal model implemented in its entirety as a system of chemical reactions. The *chemical neuron* (CN) consists only of micro-reversible reactions following mass-action kinetics. In this chapter, I will discuss the question of autonomy in more detail and propose a concrete list

of requirements which need to be met in order to claim that the system is autonomous. I will demonstrate that it is possible to design models which can implement both the neuronal functions, as well as the learning algorithm. Extensive simulations will be presented, showing that this model is capable of learning in a similar fashion to a spiking neuron with STDP. Moreover, I will demonstrate how this system relates to other previously published learning frameworks for artificial single-celled organisms, and propose a more biochemically plausible version of the CN which could potentially be synthesised in the context of synthetic biology.

In the last chapter of the thesis, I will introduce the notion of DNA computing in more detail. In particular, I am going to discuss a system implemented through DNA-strand displacement interactions, which mimics a spiking neuron. The *DNA neuron* is the first fully autonomous DNA-based machine capable of learning spatio-temporal patterns of inputs. I will demonstrate that the system implements a form of Hebbian learning similar to STDP, and can perform a variety of computational tasks including temporal correlation learning and novelty detection. The proposed model is based on the well-established and experimentally proven design motif of two-domain DNA strand displacement (Cardelli 2010). This approach proves to be a plausible strategy for designing a DNA-based learning system physically realisable in a laboratory.

1.2 Thesis contributions

The main research contributions of this thesis can be summarised as follows:

- I showed that the internal complexity of the multi-spike tempotron model is not necessary for multi-class learning. To this end, I proposed the generalised neuron model, which vastly simplifies the computation and can perform the same tasks.
- I demonstrated how weights of a spiking neuron can be implemented through the adjustment of reaction rate constants in a chemical reaction network.

- I established the criteria for autonomy of neuromorphic learning systems.
- I proposed the chemical neuron model, implemented entirely as a chemical reaction network. I demonstrated that fully autonomous Hebbian learning of spatio-temporal patterns can be achieved in a synthetic single-celled organism.
- I investigated a plausible route for implementation of neuromorphic systems in DNA-strand displacement interactions and produced a list of necessary interactions and nucleotide sequences which could be synthesised in a wet laboratory.

Chapter 2

Background

In this chapter, I will present all of the background knowledge necessary for the following research chapters. I will first describe the mechanistic details of biological neurons and relate them to models commonly used in the field of neuromorphic engineering. Multiple different neuron models will be discussed here including Leaky Integrate-and-Fire (LIF), Spike-Response model (SRM), Izhikevich neuron, and Tempotron. Having established the key neuronal models, I will proceed to introduce the framework of synaptic plasticity and unsupervised learning in spiking neural networks. Here, I will establish the concept of spike-time dependent plasticity (STDP), describe its biological justification, and discuss its different forms and interpretations typically employed in spiking neural networks research.

Supervised approaches for multi-class learning will be introduced next. First, I will focus on algorithms allowing a single neuron to perform this task. In particular, I will start with a LIF-based model: *Multi-spike Tempotron* (MST). This approach facilitates recognition of spatio-temporal patterns of activation embedded into random background noise, and allows for their classification by means of outputting varied multi-spike responses. A similar outlook was also shared by Memmesheimer et al. (2014) who introduced the *Finite Precision Learning* (FPL) algorithm which will also be examined. Lastly, I will discuss other research describing learning and classification

of multiple classes of inputs in networks of spiking neurons, such as ReSuMe (Ponulak 2005) and Chronotron (Florian 2012). In the next section, I will provide a brief review of computational devices, which are specifically designed to simulate spiking neurons, or *neuromorphic hardware*.

Finally, I will discuss how computation can be implemented in non-neural biological systems. This section will include both examples from in-vivo experiments, for example in bacteria performing chemotaxis, as well as theoretical work from the field of synthetic biology. I will also discuss how single-celled organisms, both natural and artificial, can adapt to changes in the environment. In particular, I will introduce the work by Fernando et al. (2009) who showed that associative learning can be implemented in an artificial single-celled organism through gene regulatory networks. I will end this chapter by discussing perhaps one of the most promising new substrates for molecular computation - DNA-strand displacement (DSD). This method has a vast potential for enabling the construction of computational devices verifiable in a laboratory. To this end, I will introduce a framework of two-domain DSD, and familiarise the reader with the examples of DNA circuits.

2.1 Biological neuron

The human brain is perhaps one of the most complex biological systems ever discovered. It consists of a network of approximately 80 billion neurons (Schliebs and Kasabov 2014), which are the basic information processing unit in a nervous system. In real neuronal systems there exist many different types of neurons, which play various roles within bigger interconnected networks. The individual functionality of each neuron depends primarily on its morphology, structure and the size of its dendritic arbors, placement relative to the whole population, and many other factors. Many of these aspects are, however, often ignored in order to simplify the analysis in terms of their computational properties.

Figure 1 schematically depicts a biological neuron. Each neuron receives signals from a population of *pre-synaptic* neurons through its dendrites. These incoming currents are capable of changing the neuron's internal state, otherwise known as the membrane potential. When the membrane potential reaches a threshold value, an output spike is propagated down its axon. The myelin sheath, which is a collection of fatty cells wrapped around an axon, acts as an insulator and prevents the movement of ions into and out of the axon. Therefore, this element controls how well is the signal propagated down the axon, and thus ensures that a clear action potential is delivered to other neurons further down in the network.

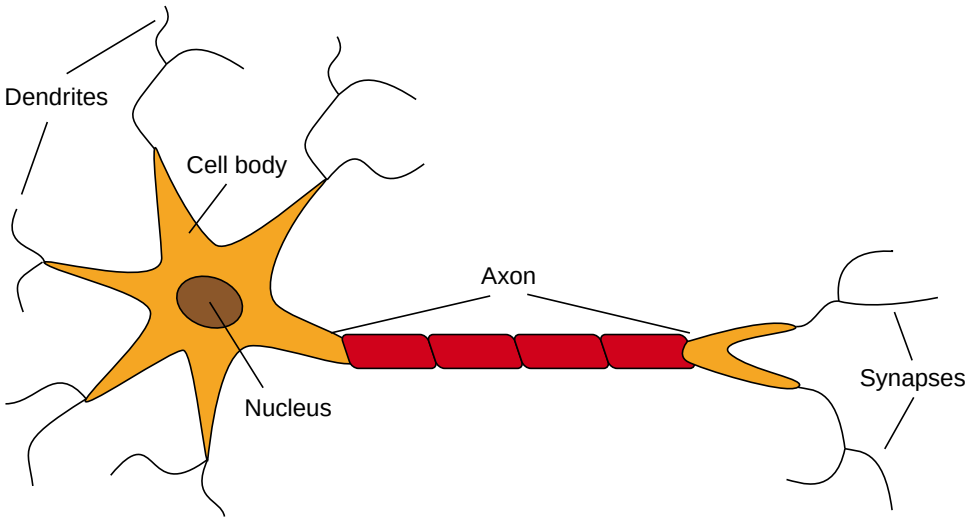


Figure 1: Schematic representation of the structure of a biological neuron.

The signal produced by the neuron is transmitted down the axon, and to all other neurons connected by its synapses. The synapse is a connection between the axon of the pre-synaptic neuron and the dendrites of the post-synaptic one. The amount of current injected to each post-synaptic neuron, and hence the effect the spike has on their membrane potential, is defined by the synaptic "weights". The weights are repeatedly readjusted to form meaningful connections during the process of learning. After the neuron has fired and reached its maximum voltage, the membrane is repolarised until it reaches a minimum value. At that time, the neuron enters a so called *refractory period*,

when it is incapable of releasing any spikes. This feature prevents the network from getting overstimulated, and provides an increased sparsity in output spike signals.

The generation of output spikes (action potentials) necessitates the neuron to be electrically polarised, i.e. maintain a potential difference across the cell's membrane. The neuron's membrane potential is fundamentally controlled by a combination of biochemical mechanisms which regulate the movement of ions entering and exiting the cell, thus allowing them to maintain a certain voltage equilibria. This is achieved through regulating the concentration gradients of different ionic channels within the cell body enclosed by the membrane. Examples of these ionics elements include sodium (Na^+), potassium (K^+), calcium (Ca^{2+}) and chloride (Cl^-) (Trappenberg 2010). One example of such mechanism are so called sodium-potassium pumps, which transport positively charged sodium ions from inside of the cell to the extra-cellular space. On the other hand, the positively charged potassium ions are transported in the opposite direction, into the cell. The membrane also has a number of ion channels, which allow for a regulated influx of sodium ions into the cell, as well as slow decay or *leakage* of potassium ions out of the cell. At the resting state, when no inputs influence the neuron's state, its membrane is polarised and maintains a negative potential.

2.2 Artificial neural modelling

2.2.1 Hodgkin-Huxley neuron model

One of the most well-known and widely recognised models of a spiking neuron is a conductance-based neuron introduced by A.L. Hodgkin and A. F. Huxley in 1952 (Hodgkin and Huxley 1952). They derived a mathematical model for a neuron by analysing the properties of action potentials in the axon from a squid. The model dynamics are based on three ionic currents driven by different elements, namely sodium

(Na^+), potassium (K^+), and a leak current. The simplified formula for each ion channel current takes a form resembling the Ohm's law:

$$I_{\text{ion}} = -g_{\text{ion}}(V - E_{\text{ion}}) \tag{1}$$

where I_{ion} denotes the current of a given ionic gate, and $\bar{g}_{\text{ion}}$ is a conductance of that channel. The expression $V - E_{\text{ion}}$ describes a net potential, where V is the membrane potential and E_{ion} is the equilibrium channel potential for a certain ion. Equilibrium potential describes a state when the channel manages to balance the flow of electricity and forces of diffusion.

The membrane potential V of the neuron is then described by the following equation:

$$C_{\text{m}}\frac{dV}{dt} = I_{\text{ion}} + I_{\text{app}} \tag{2}$$

where C_{m} is the membrane capacitance, I_{app} denotes the current applied to the system, and I_{ion} represents the sum of individual ionic current for all three channels. Thus, at any given time the I_{ion} would take the following form:

$$I_{\text{ion}} = -g_{\text{K}}\left(V - E_{\text{K}}\right) - g_{\text{Na}}\left(V - E_{\text{Na}}\right) - g_{\text{l}}\left(V - E_{\text{l}}\right) \tag{3}$$

where g_{K}, g_{Na}, g_{l} are conductances for different ionic channels, and E_{K}, E_{Na}, E_{l} are the respective equilibrium potentials for these channels.

The ionic channels are known to have voltage-dependent gates which control the flow of ions into and out of the cell. One of the key insights provided by the authors was that different gates react differently to the changes in the membrane potential. Using the data obtained from voltage-clamp experiments, they derived the following

expressions for the K^+ and Na^+ conductances:

$$\begin{aligned} g_{\text{K}} &= \overline{g}_{\text{K}} n^4, \\ g_{\text{Na}} &= \overline{g}_{\text{Na}} m^3 h \end{aligned} \tag{4}$$

where $\overline{g}_{\text{K}}$ and $\overline{g}_{\text{Na}}$ are maximum conductances for these channels, and n, m, and h are gating variables. Therefore, n^4 is the probability that a K^+ channel is open and depends upon four activation gates. On the other hand, the probability that the Na^+ channel is open is m^3, and depends on three activation gates. Additionally, there is also an inactivation gate in the sodium channel, and it's probability to be in the open state is described by h.

2.2.2 Leaky Integrate-and-Fire (LIF)

The current-based leaky integrate-and-fire neuron (LIF) is one of the most commonly used spiking neuron models. Its popularity results from the relative ease with which it can be simulated and analysed. This type of model was first introduced by Louis Lapicque, who was one of the first researchers to study excitability of nerves obtained from frogs (Brunel and van Rossum 2007). This research would later inspire the first model of the membrane potential (Blair 1932). The model is adjusted to simulate a stimulation by pre-synaptic currents and consists of a membrane potential function driven by exponentially decaying synaptic currents. Each input spike makes a contribution to the overall voltage of the neuron, and its shape over time is often described by a *bi-exponential synapse model* (Gütig and Sompolinsky 2006) (eq. 6). The membrane potential is evaluated at each time step by integrating incoming currents from N pre-synaptic neurons in the following way:

$$V(t) = \sum_{i=1}^{N} w_i \sum_{t_i^f < t} K(t - t_i^f) + V_{\text{rest}} \tag{5}$$

Where t is a current time, t_i^f is the time of the fth pre-synaptic firing of the ith

afferent, w_i is its corresponding weight, and V_{rest} is a membrane resting potential, which is assumed to be 0. $K(t)$ denotes a normalised post-synaptic potential (PSP) kernel, vanishing at $t_i > t$, contributed by each incoming spike. Its shape is defined by:

$$K(t - t_i^f) = V_{\text{norm}} \left(\exp\left(-\frac{t - t_i^f}{\tau_m} \right) - \exp\left(-\frac{t - t_i^f}{\tau_s} \right) \right) \tag{6}$$

V_{norm} normalises the PSP kernel in such a way that its maximum value reaches 1. The value of the normalisation factor is dependent on the ratio of integration time constants τ_m and τ_s following the equation below with $\eta = \tau_m / \tau_s$:

$$V_{\text{norm}} = \eta^{[\eta/(\eta-1)]} / (\eta - 1) \tag{7}$$

The LIF neuron releases an output spike whenever the membrane potential crosses a threshold ϑ. The membrane potential is then reset to V_{rest} and the neuron enters a refractory period when no extra output spikes can be elicited.

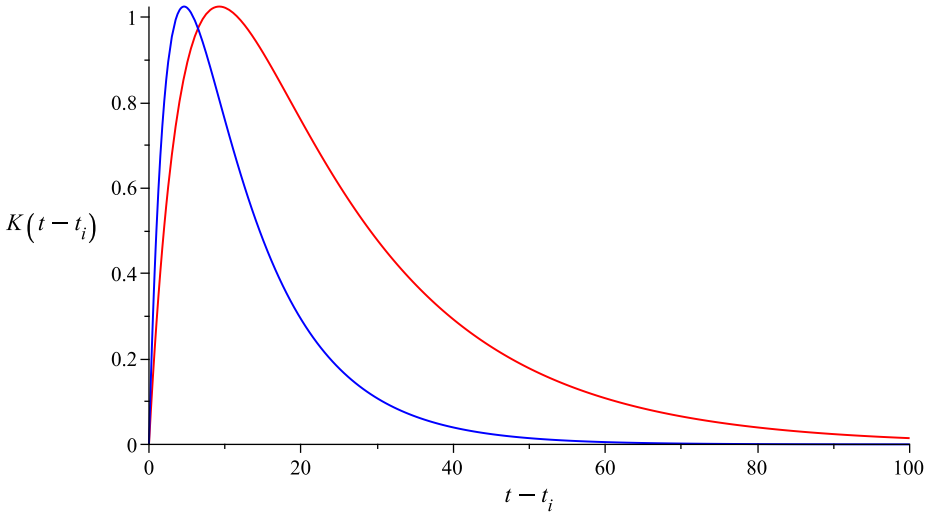


Figure 2: Shape of PSP kernel for $\tau_m = 10$ms (blue) and $\tau_m = 20$ms (red), and $\tau_s = \tau_m/4$. X-axis is defined by $t - t_i$ which is the time elapsed since the most recent input spike elicited by neuron i.

Figure 2 shows that for $\tau_m = 10$ ms the function maximum is shifted to the left. The neuron receives the maximal input earlier and the membrane potential decay is

more abrupt as well. This means that the memory of previous inputs is shorter. Gütig and Sompolinsky (2006) suggest that the τ_m parameter should be adjusted with regard to both trial time T and the number of pre-synatpic neurons N. They conclude that for τ_m significantly shorter than the mean time between inputs T/N there is virtually no temporal integration of input signals.

2.2.3 Spike Response Model (SRM)

The Spike Response Model (SRM) is a generalisation of the LIF neuron model (Gerstner and Kistler 2002b; Brunel and van Rossum 2007). Here, the membrane potential of neuron i is represented by a variable u_i. When this variable reaches the threshold from below, a spike is elicited. Importantly, in this model the spiking threshold ϑ may not be a fixed value, but also can depend on the timing of output spikes. More precisely, the spiking threshold increases when the neuron fires, thus imposing a refractory period. The membrane potential function is denoted as follows:

$$u(t) = \sum_{t_j^f < t} \eta\left(t - t_j^f\right) + \sum_{i=1}^{N} \sum_{t_i^f < t} w_i \epsilon_i \left(t - t_i^f\right) \tag{8}$$

where t_i^f is the spike time from the ith pre-synaptic neuron, w_i is its corresponding synaptic weight, and t_j^f denotes the spikes times of the post-synaptic neuron j.

The kernel functions η and ϵ in the SRM can take different forms between specific implementations. The post-spike reset kernel η models the decay of the membrane potential after firing, and is typically described in the following way:

$$\eta(t) = -\vartheta \exp\left(-\frac{t}{\tau}\right) \Theta(t) \tag{9}$$

where τ is a time constant, $\Theta(t)$ is the Heaviside function, and ϑ is the threshold value which determines the magnitude of the reset.

The kernel function ϵ describes the evolution of $u(t)$ as a function of the pre-synaptic input spikes, and can be expressed as a bi-exponential synapse in the following way:

$$\epsilon_i(t) = \left(\exp\left(-\frac{t}{\tau_m}\right) - \exp\left(-\frac{t}{\tau_s}\right)\right)\Theta(t) \tag{10}$$

where τ_s and τ_m are the time constants, and $\Theta(t)$ denotes the Heaviside function.

2.2.4 Izhikevich neuron

Another biologically plausible model of a spiking neuron was introduced by Izhikevich (2003). The main advantage of this model is that it can be tuned to simulate multiple different types of neuron dynamics such as regular spiking neurons, fast spiking neurons, or bursting neurons. The membrane potential dynamics are described by the following set of equations:

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \tag{11}$$

$$\frac{du}{dt} = a(bv - u) \tag{12}$$

$$\text{if } v \geq 30mV, \text{ then} \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{13}$$

where v denotes the membrane potential, and u is the membrane recovery variable. The parameter a describes the time scale of the recovery variable u (typical value $a = 0.02$), parameter b describes the sensitivity of the recovery variable (typical value $b = 0.2$), parameter c is the after-spike reset value of the membrane potential v (typical value $c = -65mV$), and d describes after-spike reset of the recovery variable u (typical value $d = 2$). The input to the neuron is represented by a weighted sum of the pre-synaptic

currents:

$$I = \sum_{i=1}^{N} w_i S_i(t) \tag{14}$$

The firing times of pre-synaptic neuron i are denoted by t_i^f, where f corresponds to the number of the spike. Thus, the inputs sequence from neuron i is described as the sequence of spike times $S_i(t)$:

$$S_i(t) = \sum_{t_i^f < t} \delta\left(t - t_i^f\right) \tag{15}$$

where $\delta(x)$ corresponds to the Dirac delta function with $\delta(x) = 0$ for $x \neq 0$ and $\int_{-\infty}^{\infty} \delta(x)dx = 1$. Thus the input spikes are reduced to discrete points in time.

Neocortical neurons can be divided into several types according to their temporal pattern of activation. The different types of neural dynamics they exhibit can be reproduced in the Izhikevich model by choosing specific values for the parameters a, b, c, d. The excitatory neurons can be divided into the following classes:

- *Regular spiking neurons* are the most common neurons in the cortex. Given continuous stimulation these neurons fire multiple spikes within a short burst. The temporal distance between spikes increases with each output. This kind of behaviour can be achieved by appropriate choice of c and d parameters. These dynamics necessitate high membrane potential reset ($c = -65\ mV$), and large post-spike increase of u ($d = 8$).

- *Intrinsically bursting neurons* are characterised by a firing pattern of a single burst of spikes followed by a series of individual spikes. These dynamics necessitate high voltage reset ($c = -55\ mV$) and moderate post-spike increase of u ($d = 4$).

- *Chattering neurons* fire regular bursts of spikes in close temporal proximity. In this model, this corresponds to $c = -50\ mV$ and $d = 2$, which allow for high voltage reset and relatively short post-spike increase of u.

Typically inhibitory neuron are associated with two classes:

- *Fast spiking neurons* fire high frequency spike trains. These dynamics necessitate fast post-spike recovery ($a = 0.1$).
- *Low-threshold spiking neurons* also fire high frequency spike trains, however in this case the temporal distance between the spikes undergoes frequency adaptation. This can be implemented by setting $b = 0.25$.

Moreover, this model can be also reproduce neuron types from other areas of the brain, for example thalamo-cortical neurons, as well as other types of dynamics such as oscillators.

2.2.5 Tempotron

The tempotron is a neuron model based on a leaky integrate-and-fire neuron driven by exponentially decaying synaptic currents (Gütig and Sompolinsky 2006). The membrane potential dynamics are described by the standard LIF membrane potential equation (eq. 5) with the exception of the hard post-spike reset.

Similarly to the LIF neuron, the total sum of current injected at each time step is described by the bi-exponential synapse model (eq. 6). Here, the parameters τ_m and τ_s are typically set in a way that $\tau_s = \tau_m/4$. Whenever the membrane potential crosses a predefined firing threshold ϑ, the neuron elicits a spike, and its membrane potential is reset to V_{rest} after the first output spike, which indicates class membership.

Learning capabilities

One of the functions which the tempotron can perform is to classify the "latency patterns" consisting of spatio-temporal spike trains. The neuron is given p input patterns, each of them consists of spike trains from N different input channels. Each pre-synaptic neuron is only allowed to spike once at a random time drawn from a uniform distribution between 0 and T.

Gütig and Sompolinsky (2006) showed that the tempotron model is capable of performing this task, "as long as the *pattern load* denoted by α is less than a critical value at approximately 3", where $\alpha = p/N$. This was a crucial insight since this result is higher than the fundamental capacity of $\alpha = 2$ for a single receiving rate-coded Perceptron unit (Hertz, Krogh and Palmer 1991).

Training algorithm

The tempotron model considers a classification task where each latency pattern belongs to one of the two classes: the one where a single output spike is elicited (here denoted as $\oplus$), or where the neuron remains silent (labeled as $\ominus$). The weight update function is defined as follows:

$$\Delta w_i = \lambda \sum_{t_i^f < t_{\max}} K(t_{\max} - t_i^f) \tag{16}$$

where K_t is the PSP kernel (see eq. 6), $t_{\max}$ is the time when membrane potential is in its maximum, and λ denotes the learning rate. After each training epoch the weights are either decreased by the value of Δw_i, in the case when an unwanted spike was elicited ($\ominus$ pattern), or increased when no spike was elicited for a pattern in $\oplus$ class.

Formal requirements

The temporal summation of input spikes requires the τ_m parameter to be adjusted with regard to the average time between input spikes T/N. Where T is a total time of a trial, and N denotes the number of synapses connected to the neuron. The optimal value of τ_m for N = 500 was determined to be $\sim$ 10 ms (Gütig and Sompolinsky 2006). The range of feasible values of τ_m changes with different pattern load. The learning rate λ is chosen in the following way: $\lambda = 3 \times 10^{-3} T/(\tau_m N V_{\text{norm}})$, where N is the number of pre-synaptic neurons, T is duration of the trial, τ_m is the time integration parameter, and V_{norm} is the normalisation factor of the PSP kernel (7)(Gütig and Sompolinsky 2006).

2.3 Synaptic plasticity

Taylor (1973) has demonstrated that on average biological synapses are updated as a function of the relative spike time compared to the output neuron's activity. Spiking neurons are known for their capacity for unsupervised learning of patterns in time (Gütig 2014), usually in a form of spike-time dependent plasticity (STDP) (Feldman 2012; Maass 1996). In this scenario, a neuron is not given any feedback. The learning depends solely on the statistical distribution of the input data stream. This means that typically the synaptic efficacies of neurons which spiked just before the output spike was produced will be strengthened. Reversely, the weights associated with the synapses which spiked immediately after the threshold crossing event will be decreased. This mechanism is often attributed to N-methyl-D-aspartate (NMDA) receptors, which are sensitive to changes in membrane potential of the neuron. Therefore, the synaptic weight update size is determined by the fraction of NMDA receptors open at the time of an input spike. Nevertheless, there are different interpretation of the role of these receptors. For example, Gütig (2016) motivates the design of his aggregate-label learning on the same mechanism.

The STDP function is typically described as follows:

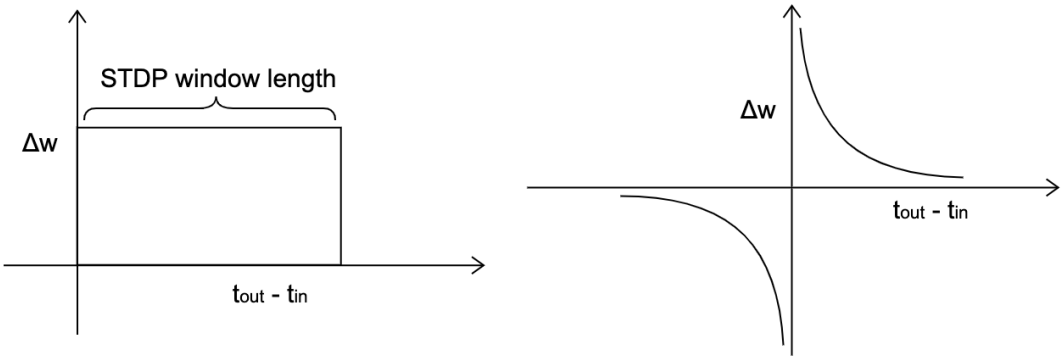
$$\Delta w_j = \sum_{i=1}^{N} \sum_{t_i^f < t} W\left(t_i^f - t_j^f\right) \tag{17}$$

where function $W(t)$ determines the shape of the learning window, and t_i^f and t_j^f denote the spiking timing of the input neurons and the output neuron respectively.

The researchers typically distinguish two types of weight update functions, namely $additive$ ($w_i = w_i + \Delta w_i$) and $multiplicative$ ($w_i = w_i \Delta w_i$) STDP. Both methods have their advantages. The additive STDP may be simpler to compute, however has been show to be unstable in deeper feed-forward networks, due to the cumulative nature of the update rule. Since the current weight value is not taken into consideration

when calculating a subsequent update, this rule tends to develop weight drifts (Gerstner and Kistler 2002a). On the other hand, in multiplicative STDP the weight update for each spike event is calculated in a way that considers the current state of the synapse. Therefore, this rule is less prone to develop solutions where one of the weights becomes overrepresented.

Learning temporal correlations is facilitated by the function $W(t)$ which determines the size of weight update for a particular input on the basis of its temporal distance from an output spike event. Multiple different shapes of the so called STDP learning window have been proposed over time, see fig. 3.



(a) Rectangular STDP (b) Bi-directional exponential STDP

Figure 3: Different shapes of spike-time dependent plasticity (STDP) functions.

Typically the form of the $W(t)$ function is defined by two exponential functions (Gerstner and Kistler 2002a), see fig. 3b. Zhang et al. (1998) demonstrated that this shape displays high correlation with the experimental data from real neuronal circuits. In the simplest form, this function can be describe in the following way:

$$\begin{aligned} W(x) &= A_+ \exp\left(-x/\tau_+\right) \quad \text{for} \quad x > 0 \\ W(x) &= -A_- \exp\left(x/\tau_-\right) \quad \text{for} \quad x < 0 \end{aligned} \tag{18}$$

where x is the difference between the neuron's input and output spiking times, A_+

and A_- are scaling parameters, and τ_+ and τ_- are time constants of positive and negative side of the STDP curve respectively.

2.4 Supervised multi-class learning

Some researchers tried to prove the potential of spiking neurons by demonstrating that they can perform complex tasks which cannot be performed with a single rate-coded neuron, for example recognition of multiple class of input patterns. In the SNN literature, a number of approaches exist for multi-label classification tasks in networks of spiking neurons. The purpose of this task is to classify incoming spatio-temporal patterns into different classes and to distinguish them from noise. Among the early attempts was the *Remote Supervised Method* (ReSuMe) by Ponulak (2005). It uses multiple post-synaptic neurons which, when given external feedback during training, learn to classify spatio-temporal patterns. The Chronotron learning rule (Florian 2012) allows a single LIF neuron to assign patterns into multiple classes by using precisely timed spike responses, i.e. different pattern classes are distinguished by the exact timing of the output spike released by the post-synaptic neuron. Other related approaches include the *Spike Pattern Association Neuron* (Mohemmed et al. 2012) and the *Precise Spike Driven Synaptic Plasticity* (Yu et al. 2013). Both of these allow for multi-label classification of spatio-temporal patterns, when provided with precisely timed training feedback. In contrast to the aforementioned approaches, other researchers have also proposed a number of algorithms which allow for learning multi-label classification tasks in a single neuron setup, including multi-spike tempotron (Gütig 2016) and finite-precision spiking algorithm (Memmesheimer et al. 2014). In order to understand how such a difficult task can be solved by a single spiking neuron, I am going to spend more time discussing the details of their training algorithms.

2.4.1 Multi-spike Tempotron

The multi-spike tempotron can learn to recognise and produce an appropriate reaction to specific spatio-temporal patterns called *features*, and distinguish them from distractor patterns and the background noise. According to Gütig and Sompolinsky (2006) this closely mirrors the tasks agents in nature have to perform in order to survive. For example, the life of a prey animal can depend on its ability to detect temporarily correlated clues, such as the sounds of breaking branches and a certain odour produced by an approaching predator. However, this can be difficult as these clues are embedded into the stream of background noise, and other distracting patterns of sensory data.

The challenge here is to reconcile the short time windows within which the temporal features occur, and a longer ones when the feedback arrives. This is frequently described as a "temporal credit assignment problem" and can be solved by Gütig's "aggregate-label" learning (ALL) algorithms. This means that the feedback the neuron receives at the end of each training iteration is proportional to the number of features, and does not convey any information about their timing, which clues were presented, or the total number of their occurrences.

Gütig proposes two methods to solve this problem. One is to adjust the weights based on the computation of the gradient for the critical threshold which allows to produce a desired amount of output spikes. The second approach is a correlation-based method which allows to associate the pre-synaptic input timing with the membrane potential of the post-synaptic neuron.

Model of neural dynamics

The model of neural dynamics for the multi-spike tempotron is implemented as a current-based leaky integrate-and-fire neuron model with exponential post-spike reset.

The membrane potential $V(t)$ is defined by:

$$V(t) = \sum_{i=1}^{N} w_i \sum_{t_i^f < t} K(t - t_i^f) - \vartheta \sum_{t_j^f < t} \exp\left(-\frac{t - t_j^f}{\tau_m}\right) \tag{19}$$

where the K function describes the PSP kernel (eq. 6), ϑ denotes the firing threshold, and t_{j}^f is the time of an output spike for the post-synaptic neuron j. The last part of this new membrane potential equation represent the post-spike reset of the membrane potential.

Spike-threshold-surface

One of the methods for solving aggregate-label problems proposed by Gütig was the spike-threshold-surface (STS) method. The author argues that aggregate-label learning should use a learning rule that adjusts the weights of a neuron in the direction along which its number of output spikes changes most rapidly. He points out that naive gradient-based approaches proposed at the time of publication were insufficient because of the discrete nature of spiking neurons, where the derivative with respect to the weights is non-zero only at the time of spiking. Thus, he proposed an alternative approach that is based on a continuous deterministic objective function.

In order to approximate the neuron's discrete number of output spikes, the spike-threshold-surface (STS) function was established. The function maps each theoretical threshold value to the number of output spikes produced. The critical threshold (ϑ_k^*) denotes a supremum (least upper bound of the set) at which the number of output spikes increases from $k-1$ to k.

$$\vartheta_k^* = sup\{\vartheta \in \mathbb{R}^+ : \mathrm{STS}(\vartheta) = k\}, k \in \mathbb{N} \tag{20}$$

This means that ϑ_k^* is the maximum threshold value which given a specific stimulation can produce k spikes. Hence, the neuron would produce its first spike as soon as $\vartheta = V_{\mathrm{max}}$ (the highest membrane potential achieved in a trial), and its current threshold

can be described as critical for a single spike (ϑ_1^*). If the V_{max} at this point in time lies below the ϑ: $STS(\vartheta > V_{\text{max}}) = 0$. Thus, k spikes are produced when:

$$\text{STS}(\vartheta_{k+1}^* < \vartheta < \vartheta_k^*) = k \tag{21}$$

Each individual input spike train has its unique STS shape. The algorithm proposed by Gütig (2016) allows to reshape the STS function and manipulate the position of the critical thresholds, thus achieving the desired number of output spikes. As a result, it becomes possible to replace the number of spikes, which is a discrete value, with the distance between the original ϑ and the hypothetical critical threshold which corresponds to the desired output. This means that the function becomes differentiable as each critical threshold value directly corresponds to a value of membrane potential (eq. 25) which is a function of pre-synaptic weights.

The weight adjustment is achieved by calculating the gradient of the critical threshold ϑ_k^* and applying it in the following way:

$$\Delta w_i = \begin{cases} -\lambda \nabla_w \vartheta_o^* & \text{if } o > d \\ \lambda \nabla_w \vartheta_{o+1}^* & \text{if } o < d \end{cases} \tag{22}$$

Here, o denotes the current number of output spikes, and d the desired output. The learning rate is described by λ and was fixed to be $1e^{-5}$ (Gütig and Sompolinsky 2006). For the full derivation of the ϑ^* gradient method, see Appendix A.

Correlation-based learning

The correlation-based learning algorithm is based on the correlation of the neuron's voltage and pre-synaptic activity. Gütig (2016) explains the biological rationale for this method: "In neurons, eligibility traces of this or similar form could be realised on the basis of intracellular calcium signals, which are sensitive to the coincidence of pre- and post-synaptic activity through the voltage dependence of synaptic N-methyl-D-aspartate (NMDA) receptors. It is well established that the induction of long-term

synaptic changes requires these calcium signals to reach specific plasticity induction thresholds". In this model these induction thresholds were simplified in a way that at each epoch 10% of the most eligible synapses (denoted by D_9: 9th decile) are chosen for the update.

$$\Delta w_i^{\pm} = \begin{cases} \pm\lambda_\nu & \text{if } \varepsilon_i > D_9 \\ 0 & \text{if } \varepsilon_i \leq D_9 \end{cases} \tag{23}$$

The eligible synaptic efficacies are increased (if the neuron produced less spikes than required) or decreased (if too many spikes were elicited) by the learning rate $\lambda_\nu = 1e^{-5}$. The eligibility of the ith synapse is defined by $\varepsilon_i = \sum_{t_i^f} \nu_i$, where ν_i denotes the correlation of a pre-synaptic input spike from neuron i with the membrane potential of the post-synaptic neuron $V(t)$, and K describes the PSP kernel (eq. 6):

$$\nu_i = \int_{t_i^f}^{\infty} dt V(t) K(t - t_i^f) \tag{24}$$

Formal requirements

Similarly to single-spike tempotron, the τ_m parameter has to be adjusted with regard to the average time between input spikes T/N. However, since in the case of the multi-spike tempotron more than one spike per afferent is elicited, the average number of incoming spikes per synapse also has to be taken into consideration. Hence, a safe assumption would be to adjust the parameters as follows: $\tau_m \propto (T/N) \times f$, where f is a frequency of spikes in each of input spike trains. For simplicity, it is assumed that each afferent operates within the same spike frequency: 5 Hz (Gütig 2016).

Similarly, Gütig (2016) recommends that the synaptic weights of the post-synaptic neuron to be initialised in a way that the neuron spikes with the rate of 5 Hz when driven by the 5 Hz Poisson background noise. This means that the initial random weights are multiplied by the coefficient which is negatively correlated with N - the number of pre-synaptic neurons. For example, for $N = 500$ the weights have to be multiplied by

0.022 to achieve this effect.

2.4.2 Finite-precision spiking algorithm

The finitely precise spiking neuron model introduced by Memmesheimer et al. (2014) is another example of a single neuron approach which can learn multiple classes of patterns. Here, the neuron is trained to recognise with finitely precise spike timing. The membrane potential function is based on the current-based leaky integrate-and-fire neuron model with a built-in post-spike reset. Specifically, the neuron's post-synaptic membrane potential $V(t)$ is given by the integration of exponentially decaying currents from N synaptic afferents, and a exponential reset based on the timing of output spikes normalised with regard to the threshold:

$$V(t) = \sum_{i=1}^{N} w_i \sum_{t_i^f < t} K(t - t_i^f) - \vartheta \sum_{t_j^f < t} \exp\left(-\frac{t - t_j^f}{\tau_m}\right) \tag{25}$$

where K is the PSP kernel (eq. 6), t_j^f is the time of an output spike for the post-synaptic neuron j. Similarly to the MST, the neuron undergoes an exponential post-spike reset.

In this model, the LIF neuron is trained to respond with a single spike at each time of desired output: t_d. Additionally, the algorithm has some level of tolerance with regards to the precision of the spiking. More precisely, the length of the temporal tolerance window, during which the spikes are still recognised as correct, is defined by the parameter ε.

Training algorithm

There are two cases when the weights are in need of adjusting. Firstly, when the neuron produces an output spike outside of the tolerance window, or more spikes than required. The weights are being decreased proportionally to the value defined by the PSP kernel

at t_{err}, where t_{err} is the time of an extra output spike.

$$\Delta w_i \propto -\sum_{t_i^f < t_{\text{err}}} K(t_{\text{err}}) \tag{26}$$

Secondly, if the neuron does not spike within a predefined tolerance window, the weights are being increased in the same manner, where t_{err} is the end of a tolerance window.

$$\Delta w_i \propto \sum_{t_i^f < t_{\text{err}}} K(t_{\text{err}}) \tag{27}$$

Formal requirements

Similarly to the previous models discussed in this section, the membrane potential time constant parameters have to be adjusted with regard to input spike frequency. Another requirement necessary to provide robust behaviour is $\varepsilon/\tau_m \ll 1$. The ratio of the time window length and the membrane integration time constant has to be much smaller than 1.

Precise spiking neuron

Memmesheimer et al. (2014) also proposed a variation of this algorithm: *the precise spiking neuron* which allow to train the neuron to respond to the stimuli with output spikes at exact times t_d. The membrane potential function is based on the same model of neuron dynamics as the model with finite precision (eq. 25).

In order to learn precise outputs a High-Threshold Projection (HTP) method is employed. Firstly, in order to avoid the undesired spikes and thus prevent the membrane potential from resetting at the unwanted times the firing threshold ϑ is increased in a manner preventing the neuron from spiking. Secondly, the post-spike resets are enforced at times of desired spikes: t_d. In the next step, the projection operation is applied

in order to ensure that $V(t_d) = \vartheta$ (see Memmesheimer et al. (2014) for the details). Finally, a perceptron-like learning rule is used in order to ensure that no other unnecessary output spikes are elicited.

2.4.3 Other multi-class learning approaches

Among the early attempt was the *Remote Supervised Method* (ReSuMe) by Ponulak (2005). It uses multiple post-synaptic neurons which, when given external feedback during training, learn to classify spatio-temporal patterns. ReSuMe takes advantage of both the spike-based Hebbian learning and a concept of remote supervision. The authors indicate that a spiking network trained with ReSuMe can be used to, for example, control neuroprosthetic systems.

Another popular approach is the chronotron (Florian 2012). This learning rule allows a single LIF neuron to assign patterns into multiple classes by using precisely timed spike responses, i.e. different pattern classes are distinguished by the exact timing of the output spike released by the post-synaptic neuron. This method allows a neuron to learn to classify inputs, by firing temporally precise spike trains for different inputs belonging to the same class.

2.5 Neuromorphic hardware

Nervous systems in biology operate fundamentally differently to digital computers. Simulating realistic neuron models, even at a fraction of the biological scale and synaptic connectivity, proves to be very computationally expensive on conventional computers. Furthermore, often there are additional requirements for such simulations, for example low power consumption and real-time execution (Hasler and Marr 2013). The search for suitable hardware for simulating realistic neuronal circuits led to discovery of unconventional computers which either operate according to analog principles or allow for efficient simulation of electrophysiological behaviour of neurons in digital

hardware.

Neuromorphic processors are a special type of hardware which is built with a purpose to simulate spiking neural networks. Their main advantage is that they allow for a reduction of the cost of computation. In analog neuromorphic hardware this is accomplished by limiting the amount of electricity required to compute. These neuromorphic chips, unlike von Neumann machines, do not work in a clocked mode and only send electrical pulses when necessary (Indiveri et al. 2011). Among the most recent developments in terms of analog devices are the optical neural networks (ONNs) (Xu et al. 2021). These devices allow for up to 10 trillion of operations per second, which is more than 1000 times faster than its analog predecessors.

Other processors are mixed analog-digital or fully digital chips which employ a specialised hardware to simulate networks with biologically plausible connectivity. SpiNNaker is a digital neuromorphic platform based on ARM processors, primarily used in mobile devices (Khan et al. 2008). The first generation of SpiNNaker system had 1 million cores, while the new generation SpiNNaker 2 has 20 million cores and is capable of modelling up to a billion spiking neurons and allow for 1 ms per step of simulation (Höppner et al. 2021; Plana et al. 2020; Rhodes et al. 2020). Importantly, SpiNNaker is an example of a hybrid architecture. This means that it is capable of simulating both rate-based and spiking artificial neural networks. This feature allows to harness the power of well-established deep neural network paradigms, and combine them with energy efficient feature detection provided by spiking neural networks. Other examples of fully digital chips include TrueNorth developed by IBM as part of the DARPA SyNAPSE programme (DeBole et al. 2019), or Loihi which was manufactured by Intel (Davies et al. 2018).

BrainScaleS is an example of mixed-analog-digital neuromorphic hardware system developed by research groups from the University of Heidelberg and the TU Dresden, and was funded as a part of Human Brain Project (Wunderlich et al. 2019). In addition to the analog processor, this system also contains an embedded digital processor which

allows for on-chip learning. This framework, similarly to SpiNNaker, allows for combining both spiking neurons and traditional perceptron neurons in the same experiment.

2.6 Computation in cells

There has been a significant amount of research in the area of theoretical computation in molecular systems, nevertheless most of them explore the effect of changing topology, rather than the weighting of nodes in the network (Bray 2003). There has been multiple attempts to show that arbitrary computation, such as neural networks and Turing machines (Hjelmfelt, Weinberger and Ross 1991), or finite-state machines (Hjelmfelt, Weinberger and Ross 1992) can be implemented in a chemical setup. Notably, Okamoto, Sakai and Hayashi (1988); Okamoto and Hayashi (1983); Okamoto, Sakai and Hayashi (1987) have demonstrated that switch-like behaviour and a model of McCulloch-Pitts binary neuron are realisable through cyclic enzyme systems. However, it is important to highlight that these were not autonomous systems, and they did not implement learning.

2.6.1 Chemotaxis

Single-celled organisms have been shown to demonstrate forms of learning behaviour, for example in bacteria such as Escherichia coli (Hoffer et al. 2001). The most widely studied computation mechanism may be their adaptation to the changes in the environment or *chemotaxis* (Alon et al. 1999; Yi et al. 2000). Chemotaxis allows those organisms to direct their movement based on the gradient of concentration of a particular chemical compound or *attractant* in their surrounding. In that way, these organisms are able to, for example, sense and move towards the most promising source of food (glucose). $E.coli$ is a type of bacteria which uses a run and tumble method to move in its environment. Their behaviour resembles a random walk consisting of long periods of swimming in a particular direction (run), which is interrupted by periods when

the bacterium stops and rotates counter-clockwise (tumble). The organism implements chemotaxis, through modulation of the tumbling frequency. More precisely, the concentration gradient of the attractant in the environment, is negatively correlated with the tumbling frequency of the bacterium. As the concentration of the attractant increases over time, the $E.coli$ changes the directions less frequently, and therefore tends to move up the gradient.

2.6.2 Associative learning

Associative learning is typically only attributed to animals with a nervous system (Walters, Carew and Kandel 1979; Fanselow and Poulos 2005). Fernando et al. (2009) have shown that a limited case of associative learning could be implemented in an artificial single-celled organism. Associative learning is typically exemplified by the famous Pavlov's dog experiment. Here, a dog learns to associate a conditioned stimulus - the bell ringing, with the unconditioned stimulus - the sight and smell of food. After a couple of presentations of the conditioned stimulus together with the unconditioned one, the dog learnt to salivate when only hearing the bell, without the smell of the food. This type of memory is encoded via modification of the synaptic efficacies in the animal's brain.

Fernando et al. have demonstrated that such behaviour can be implemented in an artificial cellular circuit by using gene regulatory networks (GRNs) and protein kinase signalling networks. Here, the cell's memory is stored in the form of concentrations of dimerizable proteins. This work has shown that such learning paradigms could in principle be implemented in a single-celled organism, such as bacteria.

2.6.3 DNA

Deoxyribonucleic acid (DNA) is a molecule which encodes biological instructions for the development of all living organisms. It's double helix structure was first described

by Watson and Crick (1953). The long DNA chains, typically referred to as *strands*, consist of two types of material: the backbone made of deoxyribose sugar and phosphate groups. Each sugar has one of the four nucleotides attached to it. These can be either adenine (A), cytosine (C), guanine (G), or thymine (T). The DNA molecules exist in a form of a double-stranded structure which remains stable thanks to the pairing of complementary nucleotide pairs with opposite spatial orientation. The DNA molecules obey a strict nucleotide pairing, where nucleotide A can only bind to T, and C is always paired with G.

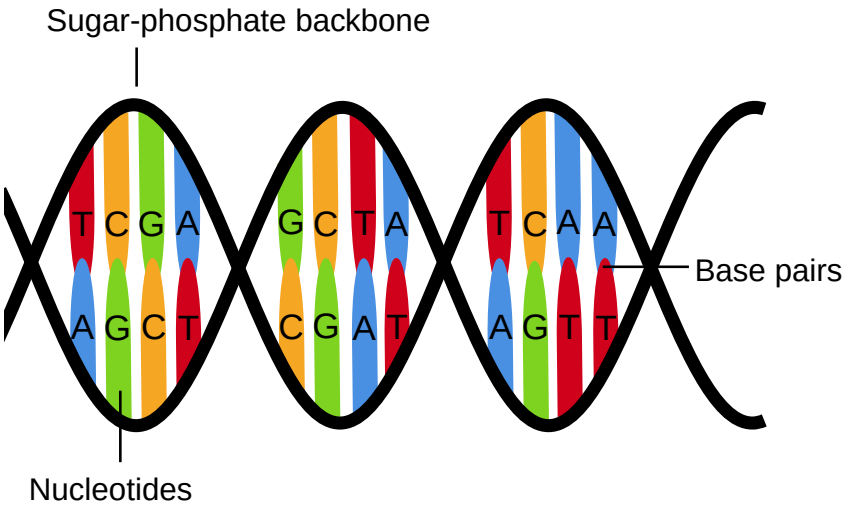


Figure 4: The double helix structure of a DNA molecule. The stability of the two strands is maintained by complementary pairing of nucleotides: adenine (A), cytosine (C), guanine (G), and thymine (T).

DNA-strand displacement

The DNA-based systems are typically analysed on two levels: sequence-level and domain-level. The former involves the study of interactions between individual nucleotide pairs, while the latter focuses on the interactions between *domains*. Here, domains are understood as sequences of nucleotides of varied length. Fundamentally, there are two types of domains which are differentiated by their length. Firstly, short

domains or *toeholds* are between 4 to 10 nucleotides, and are assumed to be able to bind and unbind from complementary strands. Secondly, long domains, which are also referred to as recognition domains, are at least 20 nucleotides in length, and are assumed to bind irreversibly.

In principle, such domains can be designed in an arbitrary manner and synthesised in a laboratory. This opens a vast field of opportunities for using these molecules as a substrate for computation. The field of DNA nanotechnology primarily focuses on constructing static structures, which exhibit certain properties, for example catalytic reactions and oscillators (Dalchau et al. 2018). Nevertheless, there has been significant advancements made in engineering systems which can work as logic gate circuits, or neural networks (Seelig et al. 2006; Cherry and Qian 2018). The design process of such systems was enabled by the framework of DNA strand displacement (DSD).

DSD is a domain-level mechanism for performing computational tasks with DNA via two basic operations: toehold mediated strand displacement and toehold exchange, see fig. 5. Importantly, one of the crucial features of this framework is its autonomy. Once the desired DNA strands are introduced to the system, the computation will proceed without the need for any further interactions from the outside.

It has been shown that these strand displacement systems are capable of universal computation (Seelig et al. 2006) and indeed that any set of chemical interactions can be realised in DSD (Chen et al. 2013). From a practical point of view, DSD systems are an interesting platform for molecular computation because they are straightforward to implement and their behaviour can also be accurately predicted (Yurke et al. 2000; Fontana 2006) using software such as Microsoft Visual DSD (Lakin et al. 2011) or other simulation platform such as Peppercorn (Badelt et al. 2020).

In order to describe the DNA molecules, I will employ a standard syntax of the Visual DSD programming language (Lakin et al. 2011). In this approach, the double-stranded molecules are denoted as `[r]`, where its upper strand `<r>` is connected to a

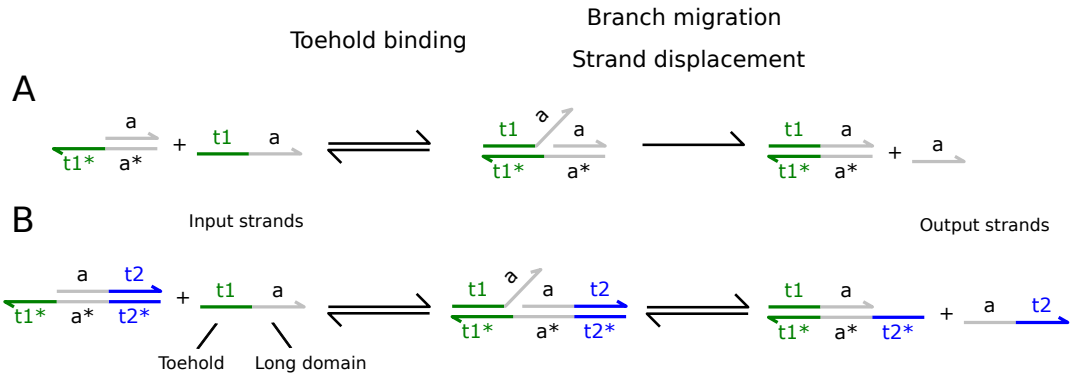


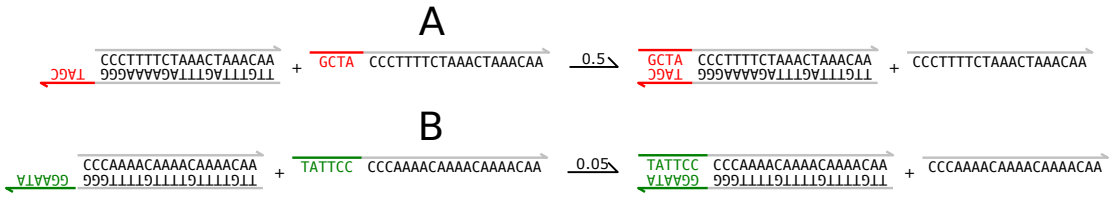
Figure 5: Example of two basic operations in DNA-strand displacement: A) toehold-mediated strand displacement, and B) toehold exchange. These two examples show that the DSD interactions can take either reversible or irreversible form, depending on the toehold domains being exposed for binding. More precisely, the long domains (here in grey) can only unbind from the double-stranded complex, once the toehold has already been attached. This distinction will turn out to be crucial for creating *sealed* DNA complexes, which become unreactive once they have served their purpose.

complementary lower strand `{r*}`. Most of the molecules in a DSD system are two-domain species, which have a single long domain and a toehold domain. Therefore, for example, an upper single-stranded two-domain molecule would be composed of a short toehold domain (annotated with a prefix `t` and an identifier `^`) and a corresponding long domain `r`. Such two-domain species would thus take a following form: `<tr^ r>`.

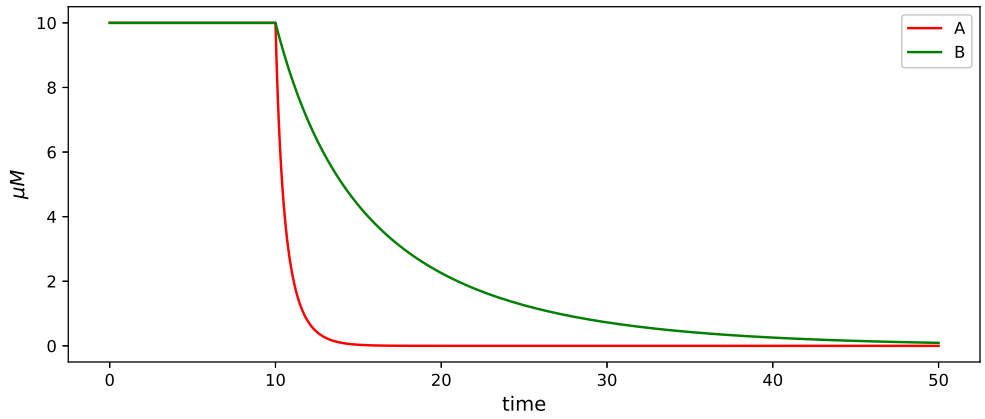
Depending on their underlying nucleotide sequence the domains can exhibit different binding properties. Thus, by choosing different nucleotide sequences it is possible to control the dynamics of the DNA strand displacement reactions. Figure 6 shows examples of toehold mediated strand displacement for two molecules with different reactivity.

The Visual DSD offers four different compilation modes, where each mode specifies a different set of assumptions for the simulation of a DSD system:

- Infinite: The rates of unbinding and migration reactions are assumed to be infinite compared to the rates of binding reactions. Therefore, strand displacement



(a) Two-domain DSD interactions.



(b) Concentration of A and B strands.

Figure 6: Examples of toehold mediated strand displacement reactions. The nucleotide sequence of the toehold domain in strand A the has a higher binding propensity than the one in strand B. This results in a quicker completion of the reaction.

is assumed to take place in a single step that merges binding, migration and unbinding. Thus, only the binding rate affects the evolution of the concentrations.

- Default: The branch migration rate is assumed to be infinite, however the rate of unbinding reactions is finite. Thus, the strand displacement occurs in two steps: first binding and then unbinding.

- Finite: The rates of unbinding and migration reactions are assumed to be fast but finite, where consecutive fast reactions are merged into a single reaction with rate τ (which is a parameter of the simulator). As a result, strand displacement is again assumed to happen in two steps: binding followed by a faster step that merges migration and unbinding.

- Detailed: In this case, binding, migration and unbinding have finite rates. The

branch migration rates are calculated from the migration rate per nucleotide and the number of nucleotides in the domain.

Two-domain DNA-strand displacement

The framework of two-domain DSD has been shown to be capable of simulating any chemical reaction. Moreover, it promised an intuitive framework for building efficient and cross-talk free implementations (Cardelli 2010; Chen et al. 2013). Here, each species present in the system is a two-domain strand, comprised of a toehold and a long domain. These species can interact with double-stranded gates which facilitate the computation. Restricting computation to two-domain strands helps to protect against unexpected interactions between single stranded species, which can occur with more complex molecules. Also, as all double-stranded structures are stable, and can only change once a single-stranded component has bound, there is no possibility for gate complexes to polymerise and interact with each other.

I will now discuss three of the fundamental structures in two-domain DSD. In order to exemplify how a catalytic reaction ($R_1 + R_2 \rightleftharpoons R_2 + P_2$) can be implemented in DSD, I will show the mechanistic details of sealed Join and Fork gates which were proposed by Chen and collaborators (Chen et al. 2013), based on an initial proposition by Cardelli (2010).

Join gate Figure 7 shows the reactions which implement a sealed two-domain join gate. The join gate is facilitated by a single double-stranded complex F_1, which allows for interaction with two-domain reactants: `{tr1^*}[r1 tr2^]:[r2 tt^]:[i].` The empty toehold domain on the top strand `tr1^` allows for binding of the first reactant (R_1), which is a two-domain species and takes a form: `<tr1 ^r1>`. Once the molecule is connected via a short domain, the branch migration of the long one occurs (`r1`). This allows for the detachment of an unreactive waste molecule, which in turn makes the `tr2^` domain available. This mechanism - toehold exchange, is the key

process by which interactions in two-domain DSD occur (see fig. 5).

As a result of the toehold exchange an intermediate form of the fuel complex is created: I_1 which has a different binding potential than the original complex F_1. Binding of the second reactant R_2 (`<tr2 ^r2>`) follows the same route, however now a different toehold domain becomes available: `tt^`. Subsequently, the branch migration of the long domain of the second reactant allows for the detachment of the translator molecules: `<r2 tt>`. This strand is later used in order to trigger the fork gate. Lastly, another two-domain species $Sjoin$ binds to the join complex irreversibly via toehold-mediated strand displacement, and as a result waste molecule W_3 is created. This seals the gate, i.e. prevents any other strands from binding to the double-stranded structure.

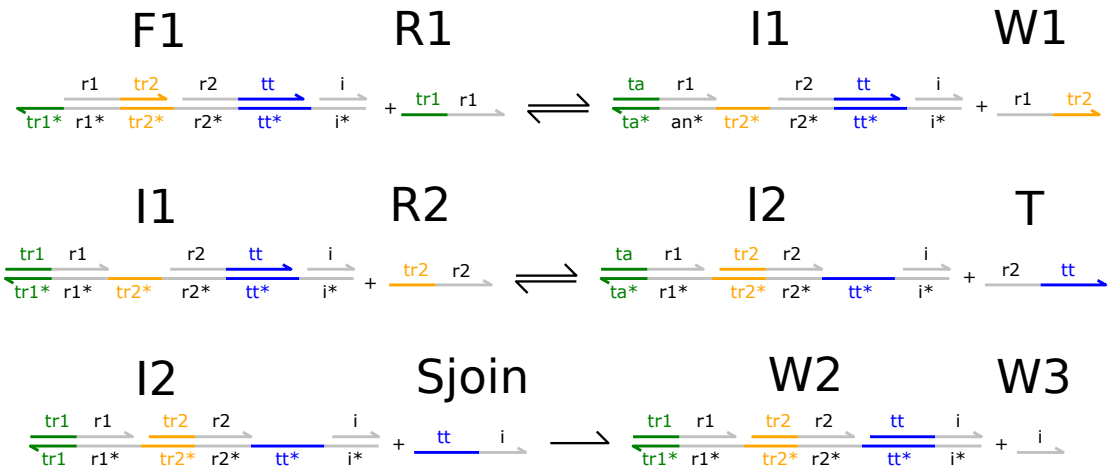


Figure 7: Sealed join gate implementing the first part of the signal modulation mechanism: $R_1 + R_2 \rightleftharpoons T$. The initial fuel complex F_1 allows for binding of the first reactant R_1 via an available toehold domain. When the first reactant binds to the complex, its structure and binding potential changes to accommodate for the second one. Once both R_1 and R_2 are connected to the double-stranded complex, the translator strand T unbinds. This molecule plays a role of a trigger for the fork gate. As the last step, $Sjoin$ binds to the intermediate complex I_2. This results in creation of two unreactive waste molecules W_2 and W_3.

Fork gate Figure 7 shows the reactions which implement a sealed two-domain fork gate. The fork gate is facilitated by another double-stranded complex F_2: `[i]:[tp2^ p2]:[tp1^ p1]{tt^*}` Similarly to the join gate, the initial gate complex starts

with one exposed toehold domain: `tt^`, which allows for a single-stranded translator molecule to bind. In turn, the gate complex releases the product strand P_1 (`<tr2 ^r2>`), which is the same as the second reactant of the join gate: $P_1 = R_2$. Next, an additional fuel molecule F_3 is used in order to release the second product P_2 (`<tp2 ^p2>`). Eventually, $Sfork$ binds to the intermediate complex I_4 creating two unreactive waste molecules W_5 and W_6.

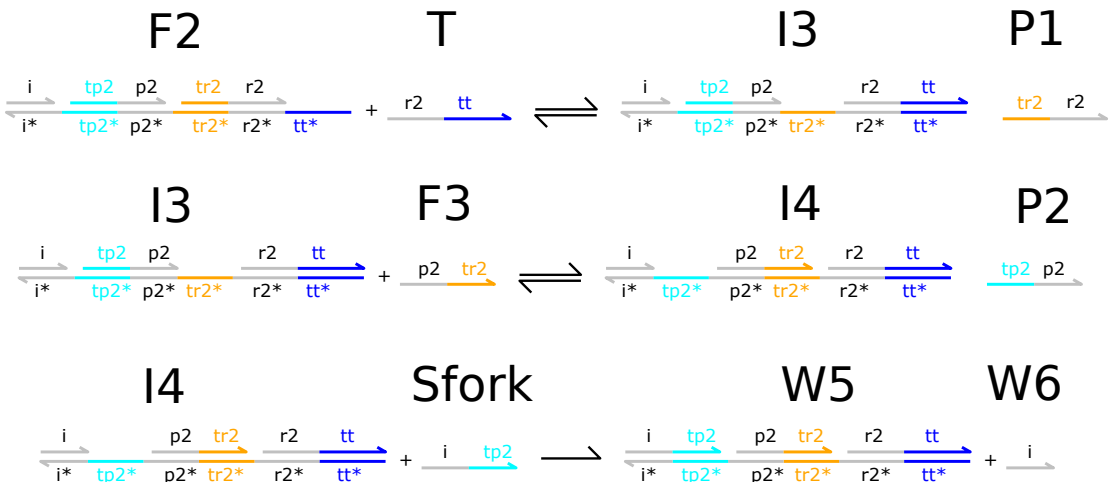


Figure 8: Sealed fork gate implementing the second part of the translator strand mechanism: T $\rightleftharpoons$ P_1 + P_2. The initial gate complex F_2 accommodates for binding of the translator strand produced by the join gate: T. Given another fuel molecule $F3$, the gate subsequently releases the product strands: P_1 and P_2. Lastly, $Sfork$ binds to the intermediate complex I_4 creating two unreactive waste molecules W_5 and W_6.

Seesaw gate The seesaw gate is another commonly used DSD motif (Qian and Winfree 2011). For example, this mechanism was used to implement weight multiplication and signal restoration in perceptron-like neural networks for winner-takes-all computation (Cherry and Qian 2018). This mechanism enables the concentrations of two two-domain strands `<t1^ a>` (input strand) and `<a t2^>` (output strand) to achieve different steady state levels based on the binding properties of their two toeholds `t1^` and `t2^`.

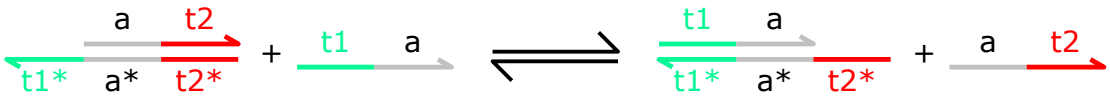


Figure 9: Seesaw gate motif in two-domain DSD. Here, the reaction is bidirectional. The concentrations of the two-domain input strands reaches a steady state depending on the amount of the reactants and backwards and forwards rates which are driven by the binding properties of the two toehold domains: `t1^` and `t2^`.

More precisely, when the gate consumes an input strand, it releases one output strand. After that, the binding properties of the gate change. The gate can now consume output strands and produce input strands. Therefore, the seesaw gate reaction is reversible.

As a result, the input species and the output species are in a constant competition for binding to the seesaw gate complex. This means that a dynamic equilibrium is found between the concentrations of these two species. The equilibrium value depends on the underlying nucleotide sequences of the toehold domains `t1^` and `t2^`, as well as their initial molecular abundances.

Chapter 3

Generalised Neuronal Model

3.1 Introduction

The neural dynamics of the multi-spike tempotron (MST) (Gütig 2016) have already been discussed in section 2.4.1, however it is important to recall some of the key observations: (i) The state of the neuron is denoted by a function $V(t)$, and is updated in discrete time. The neuron elicits an output spike when $V(t)$ crosses a set threshold value from below. (ii) The inputs to the MST neuron are provided through N unmodelled pre-synaptic channels. Importantly, the MST model includes a preprocessing step where discrete input spikes are converted into analog signals. This is implemented by a bi-exponential synapse function (see eq. 6). (iii) The update function also includes a "soft" exponential reset of the membrane potential following a spike. This sets it apart from other popular neural models, such as the leaky integrate and fire (LIF) neuron, which typically employs a "hard" immediate reset to the resting potential which is followed by a refractory period. This feature of the MST allows it to integrate inputs even after an output spikes was released, which will be demonstrated to be crucial for recognising multiple classes of patterns.

The MST is a powerful neuronal model, and has been demonstrated to solve multi-class tasks in a single neuron aggregate-label learning setup. However, it is also internally complex. In this chapter, I will examine whether or not this internal complexity is necessary for the ability of the MST to learn. In an effort to investigate this, I will systematically strip away features from the MST and check whether this impacts on its ability to learn.

Firstly, I will propose the *Generalised neuronal model* (GNM) (Fil and Chu 2020). This neuronal system can be understood a special case of the well known Spike-Response model (Gerstner and Kistler 2002b; Jolivet, J. and Gerstner 2003). In the following sections, I will rigorously explore the GNM's parameter space in order to explore crucial features of the model. The model will be tested on a variety of tasks, in order to characterise its learning capabilities in comparison to the MST and LIF neurons trained using the aggregate-label learning algorithm (ALL). This investigation will lead to the conclusion that the neuron's ability to compute is primarily dependent on two parameters: the one which determines its response to the behavioural threshold and another one which controls the decay of its membrane potential. Further investigation will show that most of the complexities of the MST neuron are not crucial for learning multi-class spatio-temporal patterns using the aggregate-label learning framework. Interestingly, I will find that there is no strict need for spiking, nor is the "soft" exponential post-spike reset of the membrane potential essential. Particular attention will be given to a parameter which controls the degree of temporal autocorrelation of the membrane potential. This ability to remember the past inputs will prove to be crucial for successful learning. Therefore, concluding that using immediate hard reset, which erases any memory of the correlation between post-spike and pre-spike membrane potentials, has a negative impact on the learning capability of a single neuron trained using ALL.

An alternative approach to training the GNM will be proposed next. Here, I will demonstrate that error-trace learning can further improve the neuron's performance.

This training algorithm includes more detailed information about the timing of the erroneous spikes, as opposed to the ALL where the neuron only receives the feedback concerning the amount of output spikes. I will finish this part of the chapter by introducing a framework for error backpropagation, which is an extension of error-trace learning. This approach can be applied to training multi-layer networks of the GNM.

Finally, I will discuss how the continuous-time model can be interpreted as a chemical reaction network (CRN). To this end, I will first show how synaptic weights can be encoded by reaction rate constants in the reaction network. Next, I will demonstrate how this neuron model can be trained, by proposing an alternative way to obtain aggregate-label feedback in non-discrete neural models. The chapter will conclude by showing that this model is also capable of multi-class learning, and discuss how the volume of the system affects the noise around the obtained solutions.

3.2 Model description

The main contribution of this chapter is the generalised neuron model (GNM), which is a parametrised family of neurons which can be tuned to display certain neuronal functions. The model can be parametrised to exhibit varying degrees of spikiness, temporal autocorrelation of the membrane potential, and hysteresis. For a detailed outline of the model see fig. 10. In this section, I am going to discuss the key parameters of the GNM, in an effort to understand what are the crucial functions allowing a model of a spiking neuron to learn.

As is typically assumed in the SNN literature, the membrane potential or the state of the neuron is described by a function $V(t)$. Similarly to the MST model, discrete

time update steps are assumed:

$$V(t) - V(t-1) = I(t) - \underbrace{(\eta\gamma R(t-1)V(t-1) + (1-\eta)\alpha V(t-1))}_{=:D(t)}$$

$$R(t) - R(t-1) = \zeta \frac{V(t-1)^h}{\vartheta_{\mathrm{B}}^h + V(t-1)^h} - \beta R(t-1) \tag{28a}$$

where $I(t) := \sum_{i=1}^{N} \sum_{j=1}^{M} w_i \delta(t_i^j - t)$ is the sum of the weighted inputs at time t and t_j^i is the time of the j-th spike of input i, and i ranges from 1 to M; $\delta(x)$ is 1 if $x = 0$ and 0 otherwise; $0 \leq w_i \leq 1$ is the weight of the i-th input. α and γ are decay coefficients of the membrane potential, ϑ_{B} denotes a behavioural threshold, and $0 \leq \eta \leq 1$ acts as a model choice parameter, ζ is a rate parameter of the Hill function, h is a Hill function coefficient, and β defines the decay rate of R. The GNM can be best understood by considering certain parameter values (see Appendix B for examples of GNM dynamics with different parameter choices.).

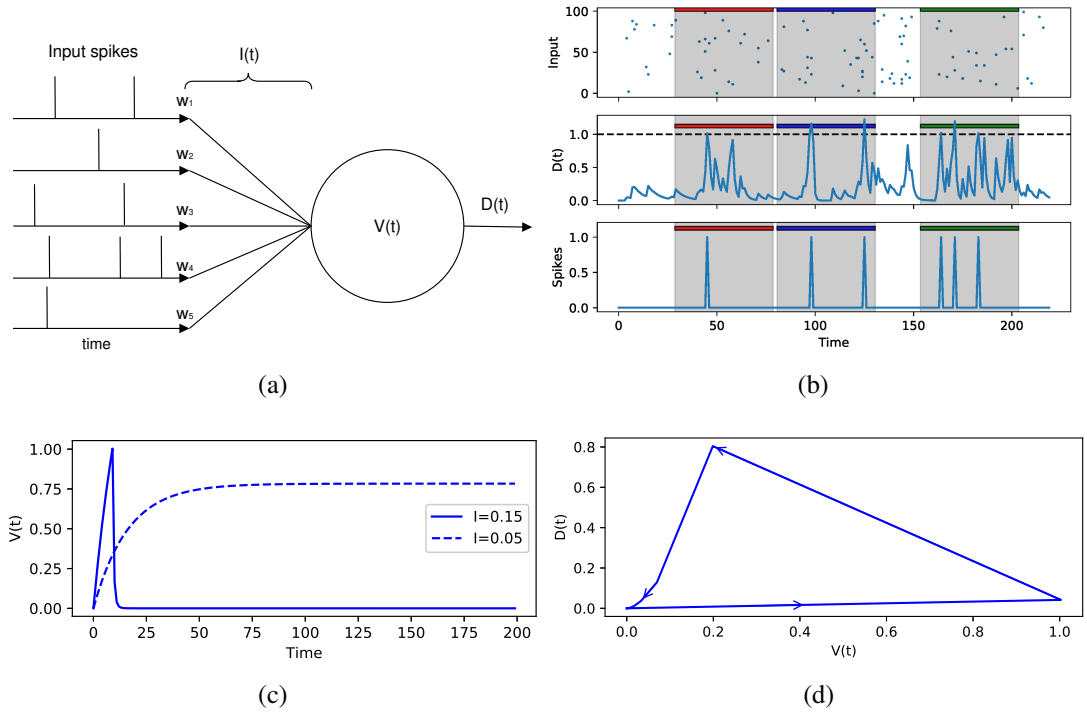


Figure 10: (a) Schematic diagram of the GNM. A single neuron has N weighted input channels, each of them receives a set of temporal input signals. (b) When the readout function reaches a threshold ϑ_{R}, then an output spike is recorded. In this example, the neuron is presented with three spatio-temporal patterns: red, blue and green. The GNM was trained to respond with a single spike to the red pattern, two spikes to the blue pattern, and three to the green pattern. Otherwise, it should remain silent during a noisy phase in-between each pattern. (c) Membrane potential as a function of time for the GNM stimulated by sub-threshold (dashed line) and super-threshold (solid line) continuous input, with parameters $\eta = 0.8$, $\alpha = 0.3$, $\beta = 0.1$. (d) Hysteretic dynamic behaviour of $D(t)$ as a function of $V(t)$ and given a single super-threshold input signal.

The parameter η controls the "spikiness" of the neuron, i.e. how responsive is the membrane potential function with respect to the behavioural threshold. In other words, it can be understood as enabling the detection of high frequencies of spiking in the power spectrum. If η is set to a positive value, then another time dependent decay rate R starts to have an influence on the membrane potential function. In neuronal terms, R can be understood as describing the number of ion-channels that only open after the membrane potential approaches the behavioural threshold ϑ_{B} and close, stochastically,

with a constant factor of β. As a result an additional decay of membrane potential is introduced when the membrane potential gets close to the threshold value. The purpose of this mechanism is to simulate a soft post-spike reset, similar to that of the MST. R is set to 0 at the beginning of the simulation. The subsequent increase of $R(t)$ is dependent on the membrane potential via a Hill-function (first term on the right hand side of eq. 28a). In biochemistry, the Hill equation describes the binding of ligands to a macromolecule as a function of the ligand concentration. Depending on the Hill coefficient h the function can display a varying degree of steepness, and thus exhibit a behaviour similar to a sigmoidal activation function:

$$f(x) = \frac{x^h}{\vartheta_{\mathrm{B}}^h + x^h} \tag{29}$$

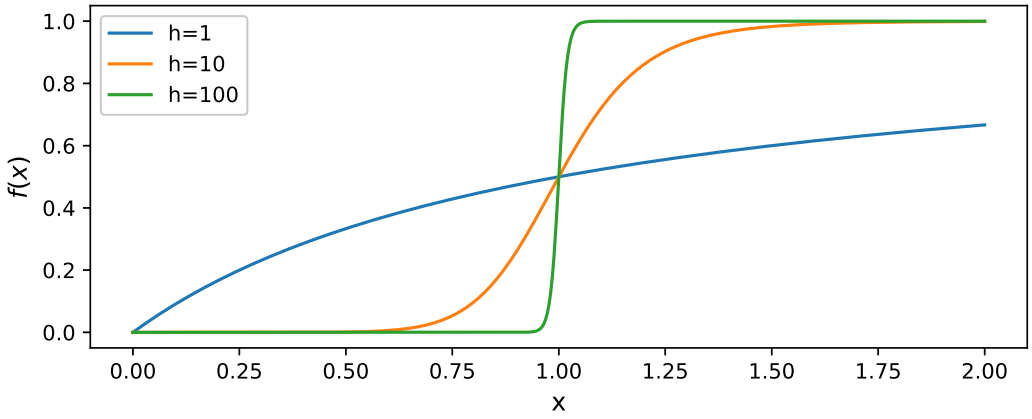


Figure 11: Hill kinetics for $\vartheta_{\mathrm{B}} = 1$ and varied h coefficient.

As the Hill function coefficient $h \to \infty$, the approximation approaches a step-function with the threshold at the behavioural threshold ϑ_{B}. Nevertheless, even when considering small finite values for h, the Hill function is close to 0 when $V(t)$ is below ϑ_{B}, and 1 when $V(t)$ is above it. Importantly, this additional decay rate R itself decays with a rate factor of β. The effect of this is that the model exhibits *hysteresis* (see fig. 10d). More precisely, given certain parameters, the membrane potential decays faster after having crossed a threshold value ϑ_{B}. The duration of this additional reset

is defined by the value of β, and will continue when the membrane potential falls back below the threshold (see fig. 12)

When $\eta = 0$ the post-spike reset of the membrane potential is disabled, i.e. the threshold does not effect the neural dynamics. Such a model lacks some of the characteristic features typically associated with spiking neurons. Setting $\eta = 0$ disables the additional decay factor R. Therefore, there is no post-spike reset of the membrane potential, as the neural dynamics become indifferent to the activation threshold. In that case, the GNM (eq. 28) reduces to a simple model of input integration with an exponential decay of the membrane potential:

$$V(t) - V(t-1) = I(t) - \alpha V(t-1). \tag{30}$$

In this basic scenario, the membrane potential only increases as a function of the input I, and constantly decays with a constant factor of α. In the extreme case of $\alpha = 0$, the neuron's membrane potential never decays. On the opposite end of the spectrum, when $\alpha = 1$, all prior inputs are forgotten at each time step. Setting $\alpha > 1$ or $\alpha < 0$ is not meaningful in the case of the discrete model. The parameter α controls the temporal autocorrelation of the membrane potential, and it will be henceforth referred to as the "memory" of the system. In other words, it can be understood as enabling the detection of low frequencies of spiking in the power spectrum. The choice of α parameter will eventually be demonstrated to be critical for the GNM's success.

Spiking neurons are typically assumed to function in discrete time dynamics. In the following sections, in addition to the discrete time dynamics (as show in eq. 28), I will also test the GNM with continuous-time dynamics. This version of the model is equivalent to the discrete implementation, and one time step corresponds to $dt = 1ms$. For the continuous-time model, the update equations are converted into proper

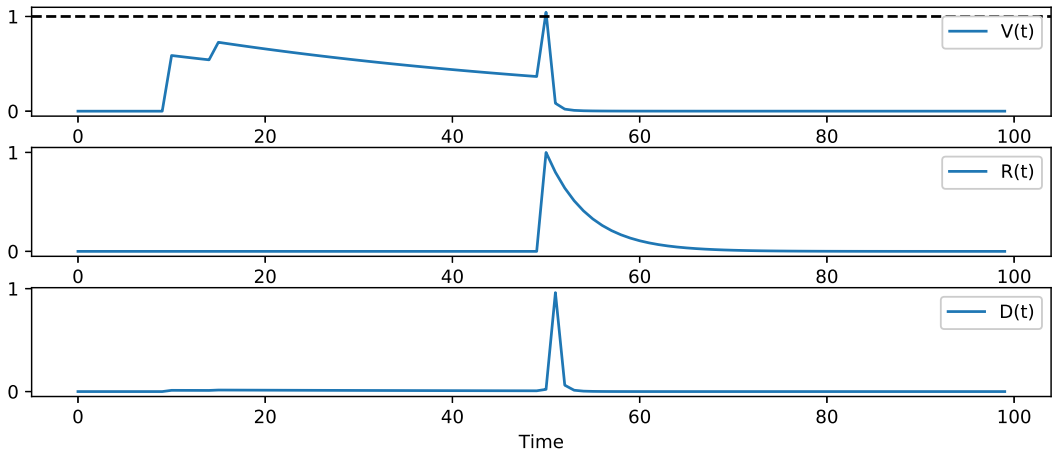


Figure 12: Example of neurons reaction in terms of the membrane potential, reset function, and the output. Here, the GNM with parameters $\eta = 0.8$, $\alpha = 0.3$, $\beta = 0.1$ and $h = 50$ is presented with three inputs, which are sub-threshold on their own. At $t = 50$, the neuron's membrane potential crosses the threshold, which in turn activates the reset and produces an output spike response. Note that the sharp output spike is possible due to the choice of a high hill coefficient h. For more examples of spike waveforms for different α and η combinations see fig. 60.

differential equations:

$$\begin{aligned}\frac{d}{dt}V(t) &= I(t) - \underbrace{(\eta\gamma R(t)V(t) + (1-\eta)\alpha V(t))}_{:=D(t)} \\ \frac{d}{dt}R(t) &= \zeta\frac{V(t)^h}{\vartheta_{\mathrm{B}}^h + V(t)^h} - \beta R(t)\end{aligned} \tag{31}$$

In the differential equation model, the parameters $\alpha, \beta, \gamma, \zeta$ become rates and are restricted to positive values, although they may be greater than 1. The η parameter remains restricted to $0 \leq \eta \leq 1$. See fig. 10a for a graphical representation of the model. When $\eta = 0$, the continuous update function of the membrane potential (eq. 31) becomes a leaky integrator and takes the following form:

$$\dot{V} = I - \alpha V \tag{32}$$

3.3 Training algorithm

I will first discuss the eligibility-based learning method similar to the one proposed by Gütig (2016) - aggregate-label learning algorithm (ALL). Here, the weights are potentiated or decayed after a period of time during which the neuron is shown target patterns embedded into noisy streams of activations. Depending on which randomly chosen patterns were presented, a target output spike count is determined. At the end of the trial, the error is then set to a negative value if the neuron spiked too many times during the trial, or to a positive value if it didn't spike enough. Otherwise the weights are not updated. Notably, this algorithm does not use any information about the degree to which the output was wrong. The only feedback the neuron receives is the "sign" of the error, and the learning proceeds by updating the weights in the following way:

$$\Delta w_i = \begin{cases} \pm\lambda & \text{if } \varepsilon_i > D_9 \\ 0 & \text{if } \varepsilon_i \leq D_9 \end{cases} \tag{33}$$

where λ denotes the learning rate which is positive when the error is positive and otherwise is negative, D_9 represents the 9th decile, or top 10% of the synapses with the highest eligibility. The eligibility of the synapse i is represented by the variable ε_i, which quantifies the extent to which the inputs from the pre-synaptic neuron i has contributed to the state of the post-synaptic neuron $V(t)$:

$$\varepsilon_i := \int_0^T I_i(t)V(t)\, dt \tag{34}$$

Momentum heuristic

Following the example of Gütig (2016) the *momentum heuristic* has been applied throughout all experiments in order to improve the speed of convergence. At each weight update step a fraction of previous synaptic change is added to the update value:

$$\Delta w_i^{\text{current}} = \Delta w_i + \gamma \Delta w_i^{\text{previous}} \tag{35}$$

where γ is the momentum parameter (in all experiments discussed in this chapter γ is set to 0.2).

3.4 Aggregate-label learning

In this section, I will demonstrate that the discrete-time GNM can learn a single spatio-temporal pattern given aggregate-label feedback. In this work, a *pattern* is understood as a temporal sequence of M binary strings of length N. In all of the following experiments, these values are set to $M = 50$ time-steps and $N = 100$ pre-synaptic neurons respectively. These patterns are generated randomly by drawing each of the bits from a Bernoulli distribution with $p(1) = 0.005$. In addition to randomly generated but fixed patterns, the neuron is exposed to a stream of *noisy background activity*. This random activity is generated in the same way as the patterns, however it is randomly produced at each individual time-step. Therefore, the statistical properties of noisy background activity and patterns are identical in this setup.

It is worth noting that unlike the MST, the GNM does not have discrete output spikes. Thus, in order to interpret the output of the GNM an arbitrary *readout threshold* value needs to be set ϑ_{R}. The threshold values used in the following sections are set uniformly $\vartheta_{\text{R}} = \vartheta_{\text{B}} = 1$. The response of the GNM is determined by the number of times the membrane potential $V(t)$ (or alternatively decay $D(t)$) crosses ϑ_{R} from below within the duration of the pattern, see fig. 10b. This number is used to indicate

class membership. In the case of learning just a single spatio-temporal pattern, the membrane potential is required to cross the threshold exactly once during the duration of this pattern. All of the experiments discussed below have been carried out in Python using software written specifically for this project [1].

Firstly, the performance of the GNM is tested on the task of learning a single pattern using the ALL algorithm (see section 3.3). In this task, GNM needs to respond with exactly one spike in response to each presentation of the trained pattern and should stay inactive otherwise, i.e. if presented with noise. The neuron is presented with 500s of a randomly generated stream of noise with the target patterns inserted into it at random times. The number of such pattern insertions is chosen at random from a uniform distribution within a range from 0 to 10. During each trial, the GNM receives a delayed feedback indicating whether it has released too many or too few spikes, and the weights are adjusted accordingly. This algorithm is repeated for 60000 epochs and the learning rate was set to $\lambda = 0.0001$, which proved to be sufficient for successful learning.

After training, the GNM should remain silent when presented with noise, but should respond to the patterns. In practice, however, the GNMs will never learn perfectly. It is bound to eventually produce an erroneous output when given continuous stimulation. In an effort to quantify the learning quality of the neuron, the number of time steps before the GNM failed is recorded. I will henceforth refer to this metric as the *noisy performance* (np) and it will be used as an indicator for the quality of the GNM learning. A high value of np indicates that the neuron can distinguish patterns from the background noise well.

In order to evaluate the model's ability to learn, 1681 GNMs with different parameter settings were generated. These models differ in their setting of α and η parameters. Both of these parameters were varied in a range from 0 to 1, and 41 different values were sampled uniformly. The other parameters are kept fixed at: $\beta = 0.3$, $\zeta = 1$, $\gamma = 1$, and $h = 50$ in order to enable the GNM to exhibit a post-spike reset closely resembling

[1] The details of the software implementation are available on https://github.com/jf330/HystNeuron/

that of the MST. Each of these models was tested on 100 generated patterns, and the average np across these tasks was recorded. Figure 13 shows a qualitative landscape of this parameter space. In order to better understand this result, I will now discuss particular parameter choices and provide a justification for their performance and the emergence of the line of optimal performance. The presented results can be interpreted in terms of the well-established plasticity-stability dilemma (Mermillod, Bugaiska and Bonin 2013), where for successful learning the neuron is required to display both plasticity for obtaining new knowledge, but also stability in order to avoid the forgetting of previous knowledge.

Let's first examine the case of $\alpha, \eta \approx 0$. As can be seen from the figure, the performance at the top left corner is rather poor. The failure to perform can be attributed to the fact that in this region the decay of the membrane potential $V(t)$ is low or nonexistent, and thus the GNM integrates over all past events. This means that the membrane potential remains in a permanent super-threshold state.

As noted before, at $\eta = 0$ the neural dynamics of the GNM reduce to $V_i(t+1) = V(t) + I - \alpha V_i(t)$. Thus, the model's dynamics are not affected by the behavioural threshold ϑ_{B}. As a result, it lacks one of the features typically associated with spiking neurons - the post-spike reset of the membrane potential. Regardless, allowing more leakage by increasing α while keeping η at 0 improves the performance and allows for successful classification. This shows that the system is sensitive to the α parameter. Adjusting α from 0.05 to just 0.08 results in the mean noisy performance increasing from approximately 0 to the globally best result. As α is further increased towards 1, the performance drops. This can be attributed to the lack of temporal integration.

When η is increased above 0, the behavioural threshold ϑ_{B} starts to influence the neuronal dynamics. At the extreme case of $\eta \approx 1$, the threshold dynamics dominate the neuron and the membrane potential leak becomes negligible. Thus, the membrane potential is constrained to a small range of values, which in turn make learning impossible. Figure 13 shows that the best performing models concentrate along a fuzzy line

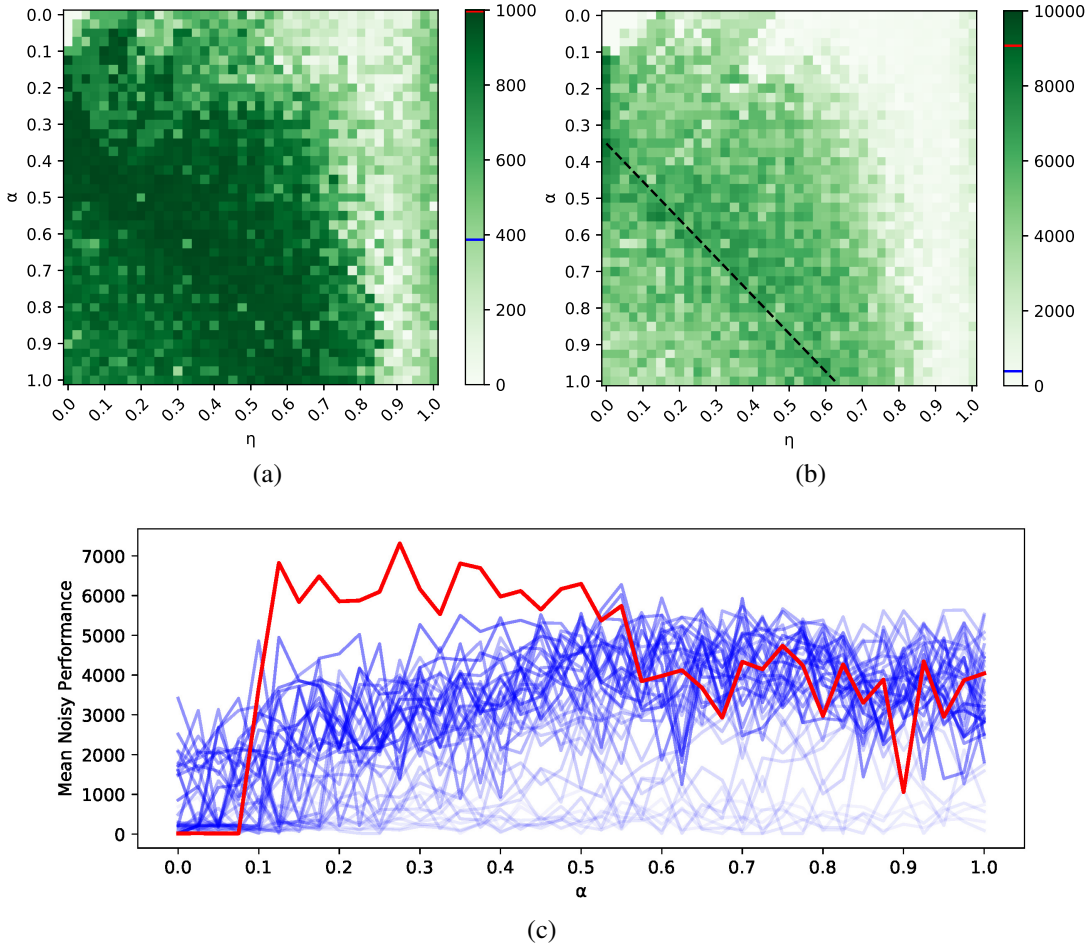


Figure 13: Mean noisy performance averaged over 5 training iterations as a function of α and η. The values reported in this figure represent the average number of time steps a neuron was able to withstand without eliciting an erroneous spike. The simulations were capped at (a) 1000, and (b) 10000 time steps, and trained for 60000 epochs. The performance of a corresponding MST neuron averaged at approximately 377 time steps, it is marked on the colour bar in blue, while the red mark indicates the maximum GNM performance on each heatmap. In (b) the dashed line shows the *optimal line* (see section 3.4 for further explanation). (c) Noisy performance as a function of α. Here, the red line depicts the performance of models with $\eta = 0$, and the blue lines show 40 other settings of η.

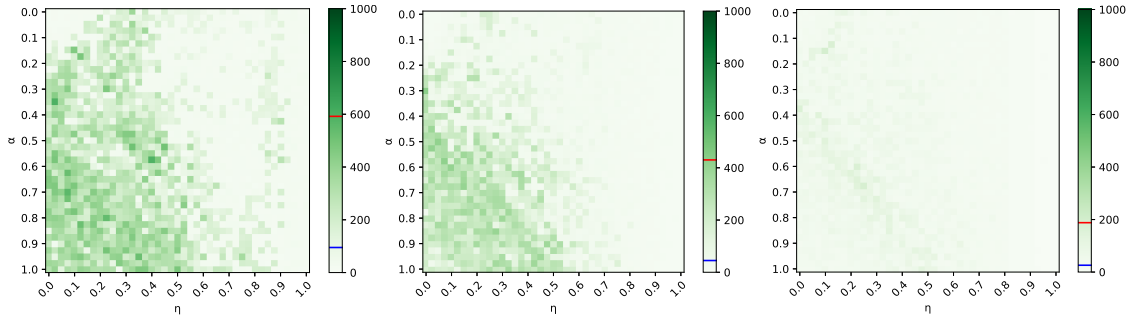
of combinations of α and η. This line will henceforth be referred to this as the *optimal line* (see fig. 13b). However, relative to the non-spiking case of $\eta = 0$, the performance along this line does not significantly increase for other models with $0 < \eta \ll 1$. Indeed, the globally best performance is achieved with parameters $\eta = 0$ and $\alpha \simeq 0.3$. Therefore, it seems that, at least for the case of a single pattern, introducing spikiness does not bring any visible benefits.

There is a particular reason for the existence of the optimal line. As can be seen from eq. 28a, the membrane potential decay is effectively reduced by $(1 - \eta)$. This means that the optimal line may be a consequence of there being an optimal value for the leak parameter α. If this optimal value is given by $\alpha = \alpha^*$ and the actual value of α is set to $\alpha' > \alpha^*$. A suitable choice of η which satisfies $(1 - \eta) = \alpha^*/\alpha'$ could offset the non-optimal choice of α to the optimal value. This effect plausibly explains the observed optimal line in the parameter space. Apart from this correction of the decay parameter, an increased η does not seem to bring any benefits in terms of performance. Therefore, the temporal autocorrelation of the membrane potential is revealed to be the crucial parameter for learning.

Nevertheless, it is important to emphasise that the conjecture that there exists an optimal value for the leak parameter α is conditioned on the fact that the patterns presented to the neuron have a fixed duration and spiking frequency. Indeed, this optimal α would necessarily change for patterns with different firing statistics. While determining the optimal value for other types of patterns would be interesting, the research conducted in this chapter focuses on learning input features constructed in a way that was originally proposed by Gütig (2016), in order to provide a more fair comparison to the MST model.

3.5 Multi-class aggregate-label learning

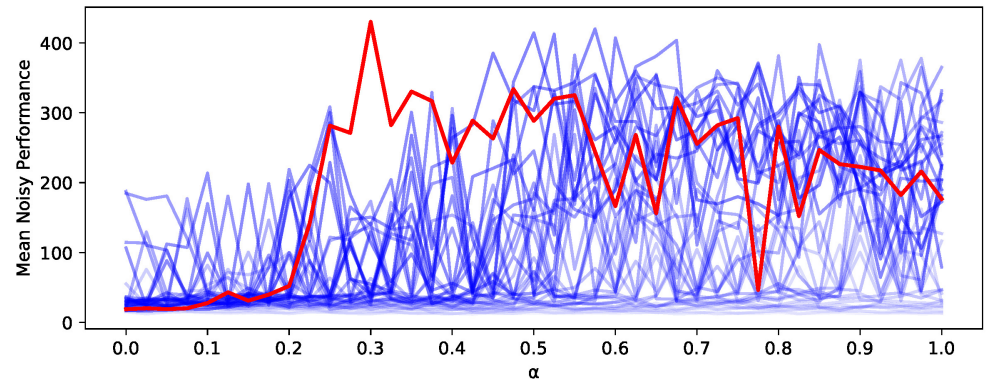
The MST is an example of a single neuron approach which can learn to recognise multiple patterns, as well as multiple classes of patterns. The MST can be trained to recognise a set of patterns with a single spike, another set of patterns to which it responds with two spikes etc. In this section, I will test whether the GNM can perform the same task. Similarly to the single-pattern case, the GNM is considered to "spike" n times if the membrane potential crosses the readout threshold ϑ_{R} from below exactly n times. The neuron will be exposed to two, three, and four different randomly generated patterns, and will be required to indicate their class membership by releasing a different amount of spikes within the duration of each pattern. Thus, if a pattern belongs to class 1 the neuron should react with one output spike, if it belongs to class 2 two spikes should be elicited etc. The patterns themselves are generated in the same way as described in section 3.4.



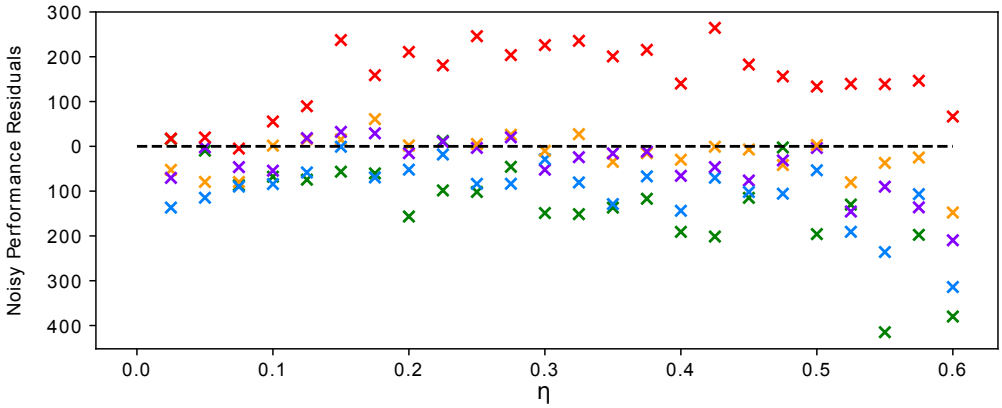
(a) (b) (c)

Figure 14: Mean noisy performance averaged over 5 training iterations as a function of α and η (same as fig. 13), for the task of learning (a) two, (b) three, and (c) four classes of patterns respectively. The performance of the MST is indicated by the blue mark on the colour bar, while the red mark denotes the maximum GNM performance on each heatmap.

The multi-pattern case shows a qualitatively similar pattern in parameter space as the single pattern case (see fig. 14). Again, a line of optimal performance emerges in the bottom left corner of the heatmaps. Nevertheless, the noisy performance of the GNM



(a)



(b)

Figure 15: (a) Mean noisy performance as a function of α for same data as in fig. 14b - three classes of input patterns. Similarly, the red line indicates the models with $\eta = 0$, and the blue lines show 40 other η settings. (b) Noisy performance residuals for each of 5 training simulations (indicated by different marker colours) as a function of η in range from 0.025 to 0.6. Each point represents the performance difference between the best performing α for any given $\eta > 0$ and the model with $\eta = 0$ which performed best. Thus, the deviation from the dashed line indicates the difference in performance in comparison to $\eta = 0$. Here, the negative values indicate that the best training for a particular η was worse than $\eta = 0$. Notably, only for a single set of input patterns the GNM with $\eta = 0$ was suboptimal. This points to the conclusion that the variation in performance depends primarily on the random seed used for pattern generation.

shows signs of decline as the number of pattern classes increases. In the case of the most difficult tasks with 4 different pattern classes, the GNM produces an erroneous spike after approximately 200 time-steps on average (see fig. 14c). Across different parameter choices tested, there does not seem to be a typical mode of failure for the GNM. The neuron is equally likely to fail on the noise, as it is on the trained patterns.

Similarly to the case of single pattern learning, there does not seem to be any apparent benefit in setting the model choice parameter above $\eta = 0$. Although for some of the generated patterns, the performance of the GNM with $\eta > 0$ was better, there was no consistent best value of η and the best model with $\eta = 0$ always performed on par with the globally best result (see fig. 14b and fig. 15). For a better understanding of this phenomenon, see the noisy performance residuals graph for three input patterns in fig. 15b. Residual graphs for learning of two and four spatio-temporal patterns can be found in Appendix C.

The residuals are defined as a performance difference between the best α for any given η and the best performing model with $\eta = 0$. In other terms, for each column in a heatmap, the best performing row is selected and then compared to $\eta = 0$. This means that negative values of the y-axis, below the dashed line, indicate that the best performer for a particular η was worse than that of $\eta = 0$, for a given set of feature patterns. Most of the points lie under the dashed line, which indicates that the models with $\eta = 0$ are indeed competitive. The GNM was shown to perform reasonably well on the multi-label classification task. The performance is competitive with the more computationally complex MST model (marked by the blue line on the colour bars in fig. 13 and 14). Notably, the simplest "non-spiking" case of the GNM ($\eta = 0$) is sufficient for solving these multi-class tasks.

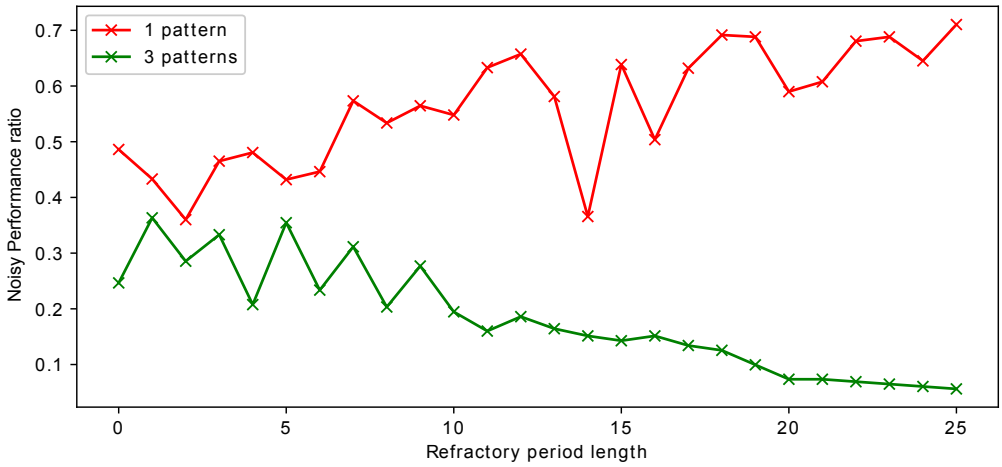


Figure 16: Noisy performance ratio between the GNM with $\eta = 0$ and an equivalent LIF neuron on the two tasks: learning one and three classes of patterns. Here, the noisy performance ratio is defined as the average noisy performance of LIF neuron divided by the result produced by the GNM trained with the ALL algorithm. Both neuron models have been trained for 60000 epochs. The following parameters were used for the GNM: $\alpha = 0.3$, $\eta = 0$, $\beta = 0.3$, and LIF: $\alpha = 0.3$ and varied length of the refractory period.

3.6 Comparison to other neural models

The GNM can be contrasted against other well-known neuronal models, such as the LIF neuron. The LIF neuron behaves similarly to the GNM model with $h = \infty, \zeta = 1, \gamma = 1/\eta$ and $0 < \eta < 1$ up to reaching the behavioural threshold ϑ_{B}, including the post-spike reset. However, after the reset, the LIF neurons typically undergo a deterministic refractory period. During this time the neurons become insensitive to further inputs. Note, that the GNM does not allow for simulating this type of refractory period. In an effort to validate the results presented in the previous section, I will compare the GNM's performance to the LIF neuron trained for multi-label classification. The GNM and the MST differ from the LIF neuron in two significant ways: (i) These models don't have a refractory period and (ii) the membrane potential reset function following an output spike is exponential, rather than a hard reset to the resting potential.

Figure 16 shows the comparison of the ability of the LIF neuron to recognise patterns with the GNM assuming $\eta = 0$ and $\alpha = 0.3$. The same parameter $\alpha = 0.3$

was used for the leak rate of the LIF neuron, and the length of the refractory period was varied from 0 to 25. Similarly to previous experiments, the two neural models are contrasted by calculating the noisy performance ratio, which is the average noisy performance of LIF neuron divided by that of the GNM trained on the same task. In the case of a single pattern, the LIF neuron performed comparably to or slightly worse than the GNM. It can also be noticed that the performance increases with the length of refractory period, see fig. 16. For the task involving two patterns, however, the LIF neuron performed worse than the GNM. On the contrary to the single pattern case, now the performance drops with the increase in refractoriness, thus imposing such a period of forced inactivity seems to limit the neuron's ability to perform the multi-spike response.

It's important to emphasise that the LIF neuron with no refractory period is identical to the GNM. The only exception being the hard post-spike reset. Nevertheless, the presented results suggest that the LiF neuron still performs worse on the task of multi-label classifications. This observation leads to the conclusion that the hard reset is responsible for the decreased performance of the LIF model. Importantly, this is consistent with the earlier conjecture that the temporal autocorrelation of the membrane potential is the crucial parameter for learning.

3.7 Alternative learning approaches

In the previous sections, I have demonstrated that the GNM can indeed perform comparably to the MST. Nevertheless, the comparison might have been unfair because a single parameter setting of the MST was compared to a large number of simulations of the GNM. In this section, I will now more rigorously validate the performance of the GNM relative to the MST, as well as the GNM trained using an alternative learning algorithm.

To this end, 50 training simulations for the task of recognising two patterns were

conducted. Two sets of fixed parameters were chosen for the GNM: $\eta = 0.0$ and $\alpha = 0.3$ and MST: $\tau_m = 20$, and $\tau_s = 5$ respectively. Figure 17 shows that the GNM outperformed the MST, trained with ALL, in most of the simulations (47 out of 50).

3.7.1 Error-trace feedback learning

The GNM's performance can be improved by applying a different training approach which utilises precise information about the timing of erroneous spikes. This method encompasses information about when and which feature patterns should have been recognised. This algorithm will be henceforth referred to as error-trace learning (ET). The error trace is defined by the values of the integral at the time when each of the patterns was presented, rather than after an entire episode of patterns and noise sequences. This way, the algorithm utilises more detailed information about which channels caused erroneous spiking for each pattern separately. More precisely, if the neuron is supposed to cross the readout threshold exactly twice during the presentation of the pattern unlike in the previously described ALL algorithm, here the neuron will be penalised if any of the spikes occurs outside of the duration of the pattern. The error is calculated for each individual synapse by correlating its inputs with the error trace:

$$\begin{aligned} \Delta w_i &= \lambda \varepsilon_i \\ \varepsilon_i &:= \int_0^T I_i(t) E(t)\, dt \end{aligned} \tag{36}$$

where λ is the learning rate, and $E(t)$ denotes the error trace. $E(t)$ is a function which determines the extent of an error with a greater temporal precision. More precisely, if a pattern shown between t_A and t_B produces one too many spikes than desired the $E(t)$ in this range will take the value of -1. Similarly in the case of noisy background activity, the $E(t)$ will represent the overall error produced during the whole trial. If the neuron spiked twice outside of the duration of the patterns, the error trace during the entire noisy phase will be -2. The variable ε_i can be interpreted as "error blame" of

a pre-synaptic neuron i towards the post-synaptic neuron. This variable quantifies the extent to which the pre-synaptic neuron i has contributed to the erroneous spiking of the post-synaptic neuron. The ET algorithm is compared to the ALL trained GNM and the MST neuron on the task to recognise two classes of input patterns (the same data as in fig. 14). Each of the approaches shown in figure 17 was trained on the same patterns, and the experiment was repeated over 50 training iterations. The comparison shows that the error trace learning algorithm has consistently outperformed the aggregate-label learning method. In 46 out of 50 training simulations a better result in terms of noisy performance was achieved when using the ET learning method. This shows that despite the fact that the ALL is an elegant and simple training rule, it turns out to be suboptimal.

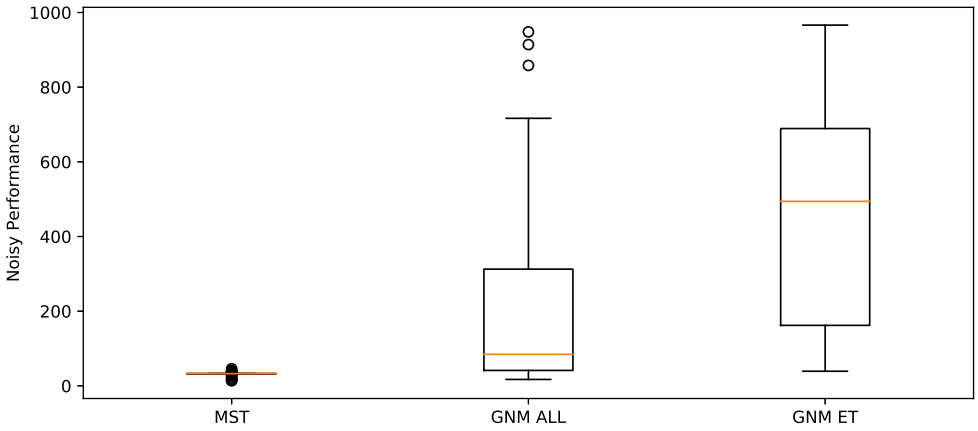


Figure 17: Comparison of the mean noisy performance measure averaged over 50 training interactions for two class recognition (the same as in fig. 14a) in MST, GNM with aggregate-label learning (ALL), and GNM with error-trace learning (ET). Each of the neuron models have been trained for 60000 epochs and the following parameters were used: GNM: $\alpha = 0.3$, $\eta = 0$, $\beta = 0.3$ (both ALL and ET); MST: $\tau_m = 20$, and $\tau_s = 5$.

3.7.2 Error backpropagation in networks of GNM

Although some of the most recent approaches to training SNNs, such as *EventProp* (Wunderlich and Pehle 2021), allow to computer exact gradients in an event-based

fashion, most of spiking neuron models can only make use of backpropagation indirectly, for example by employing surrogate gradients (Neftci, Mostafa and Zenke 2019; Tavanaei and Maida 2017). In this section, I will present the second training regime for the GNM based on error-trace - error backpropagation. The activation function of the GNM is differentiable, thus this training method can in principle be applied with no constraints. The temporal precision of the ET algorithm makes it possible to train multi-layered networks of the GNMs. The performance of the network of GNMs surpasses the learning capabilities of a single neuron trained with the ALL algorithm.

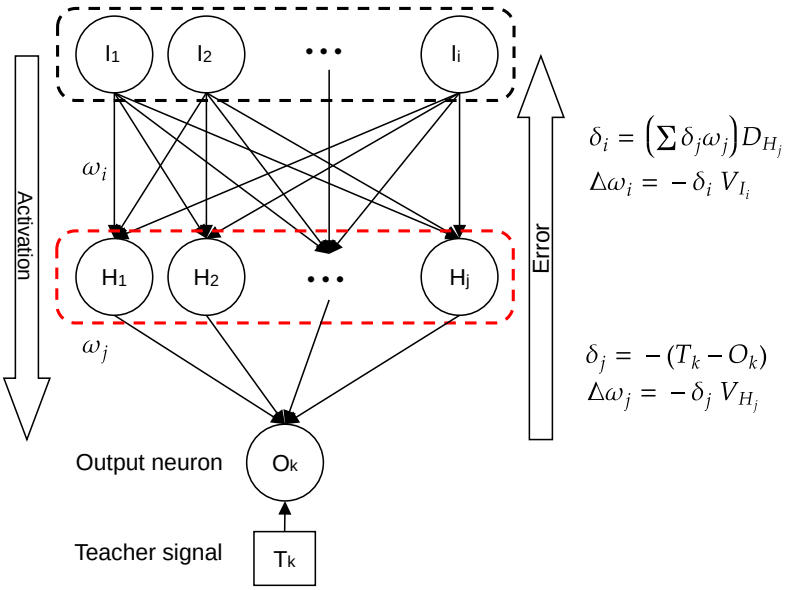


Figure 18: Diagram of a possible implementation of error backpropagation in the multi-layered network of GNMs. In this example, an additional intermediate layer of 10 hidden neurons is added between the inputs and a single post-synaptic output neuron. All of the neurons in the network have the same parameters. Each neuron in the hidden layer is connect to the input sources in all-to-all fashion. The signal from the hidden neurons is propagated to the output neuron in a form of the sum of decaying currents in the continuous form. In order for the network to learn, the error needs to be propagated back through the hidden layer and the weights need to be adjusted for both the hidden layer, as well as the output neuron. Additionally, the hidden layer neurons in the red dashed box are subject to lateral inhibition, which prevents them from learning similar subsets of the input features. This means that when one of the hidden units crosses the threshold, the membrane potential of other neurons in this layer is decreased to the resting potential.

This training approach was tested on the multi-label classification task using an architecture of layered GNMs consisting of 3 layers and 10 hidden neurons (see fig. 18). The training specification, parameters of the neurons, and the statistics of input patterns remain the same as in all other experiments presented in the previous sections. The backpropagation trained network was tested on the task to recognise three classes of input patterns. The network is trained using the algorithm described in figure 18. Importantly, the neurons in the hidden layer are subjected to lateral inhibition. Thus, when one of the hidden units crosses the readout threshold ϑ_{R}, the membrane potential of all other neurons in this layer is decreased to their resting potential. This prevents them from learning similar subsets of the input features, and as a result allows the network to achieve better performance.

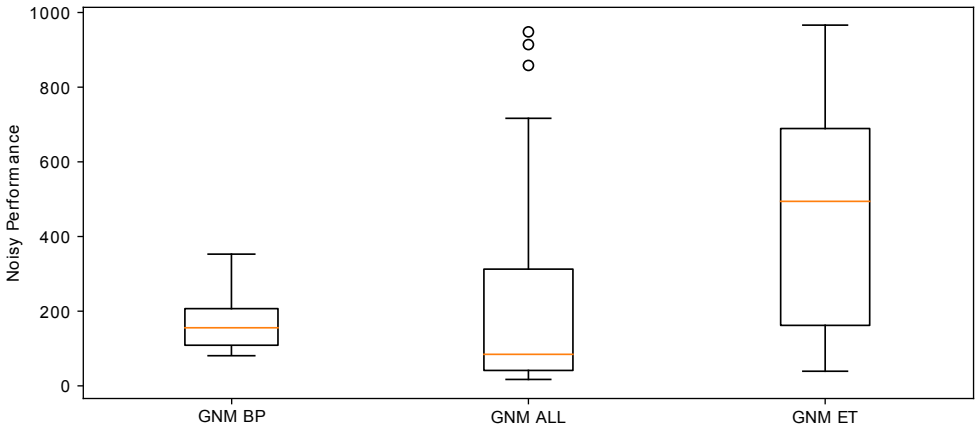


Figure 19: Comparison of the mean noisy performance measure averaged over 50 training interactions for two class recognition (the same as in fig. 14a) in GNM with backpropagation (BT), GNM with aggregate-label learning (ALL), and GNM with error-trace learning (ET). Each of the neuron models have been trained for 60000 epochs and the following parameters were used: GNM: $\alpha = 0.3$, $\eta = 0$, $\beta = 0.3$.

Figures 19 and 20 show that the multi-layered network indeed can perform this task with a high accuracy compared to the case of a single neuron. In particular, the GNM network approach performed better in terms of noisy performance metric when compared to a single neuron trained with the ALL algorithm.

Although the multi-layer networks of GNM are not the main topic of this chapter, this naive implementation shows that the GNM may have a potential for building layered neuronal populations capable of performing more sophisticated types of computation in the future.

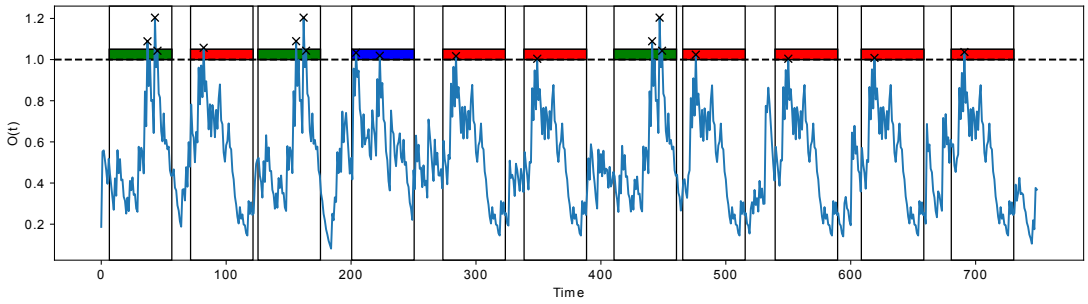


Figure 20: Example activation of the output neuron in a two-layered network of GNM, as shown in fig. 18. Here, the network is stimulated in the same way as in the previous experiments and with the same parameters. The task of the output neuron is to recognise three different classes of patterns. The output neuron has been trained for 60000 epochs to respond to each pattern with a different amount of spikes.

3.8 Continuous-time GNM and chemical implementation

The GNM model can easily be extended to the continuous-time case (eq. 31). The continuous case of the GNM model with $\eta = 0$ (eq. 32) lends itself to an interpretation as a chemical system. In that case, V can be interpreted as a molecular species which decays with a rate of α. The input I is then equivalent to N different chemical species I_i. Each of the species representing input channels decays to V with a rate of $w_i C$ and to $\varnothing$ or "null-species" with a rate of $(1 - w_i)C$, see fig. 21. The constant C determines the time-scale of the decay. Therefore, the weights are effectively implemented by the ratio between those reaction rates.

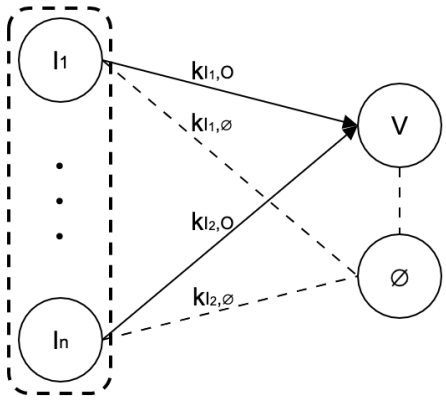


Figure 21: Diagram depicting a chemical reaction network which implements synaptic weighting in a GNM-like learning unit. Here, the weights are encoded as the ratio between the reaction rate from I_n to V, and from I_n to $\varnothing$.

The ability of this chemical system to recognise patterns can be tested by solving the differential equation 32. A crucial assumption of the differential equation model is that the number of molecules involved in the system is very large or infinite. In such a system, V would be described as a concentration. In any real system, however, the number of particles is finite. If the number of particles involved in the computation is small, the system exhibits noise around the deterministic solution of eq. 32. As a consequence, with a decrease in the number of particles, the output of the neuron becomes stochastic and produces more erroneous outputs.

Figure 22 shows how this noise affects the performance of the stochastic systems in comparison to the deterministic solution. The simulations were performed using Gillespie's algorithm and MATLAB SimBiology software package [2].

[2]The details of the software implementation are available on https://github.com/jf330/HystNeuron/

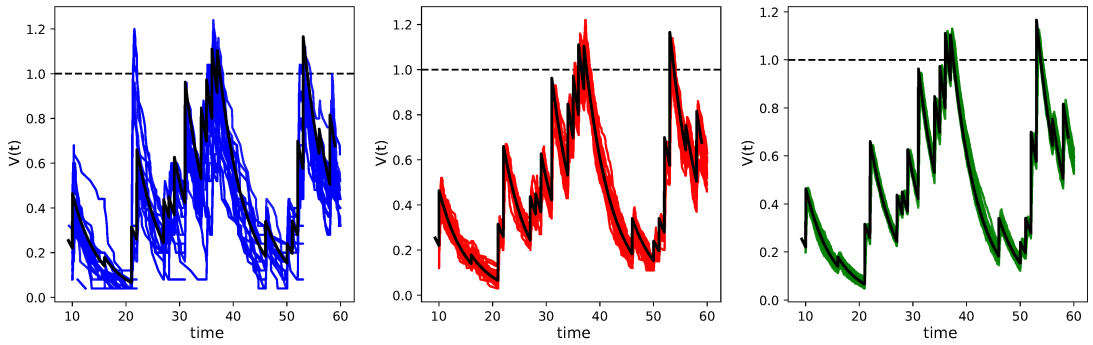


Figure 22: Example of the membrane potential trace of a deterministic continuous-time GNM neuron trained to recognise two classes of patterns (black line), and equivalent stochastic chemical reaction network simulations with inputs equivalent to 25, 100, and 500 molecules respectively. In each simulation the neuron had parameters: $\eta = 0$, $\alpha = 0.2$, $\beta = 0.3$, and C was set to 10. The pre-synaptic spikes were encoded as instantaneous increase of corresponding pre-synaptic species by N, where t_j^i is the time of the j-th spiking event of the i-th pre-synaptic neuron, and $N = 25, 100, 500$ is the number of particles that is added to the pre-synaptic species i at time t_i^j.

Various models in the SNN literature, including the MST, assume that the input channels are clocked (the system is updated in discrete time). This assumption make the system easier to simulate on conventional computers, as well as simplifies the way spikes are recorded by assigning them to a particular time step. Having interpreted the GNM as a chemical reaction network allows for alternative output recognition methods to be designed. In the continuous-time case quantifying "spiking" is based on the integral of the GNM membrane potential when it exceeds ϑ_{R}:

$$\mathcal{S} := \int_0^T \Theta(V(t) - \vartheta_{\mathrm{R}}) V(t)\, dt \tag{37}$$

where Θ is the Heaviside function, ϑ_{R} represents a readout threshold, and T is the duration of the trial. The error is calculated by the difference between the actual and the desired output $\mathcal{S}$. Using this error the training proceeds following the ALL algorithm (see section 3.3).

Training the model in continuous-time yields qualitatively the same results as the discrete-time case. Figure 23 shows the feasibility of this interpretation by showing the

behaviour of a continuous-time version of the GNM trained to recognise two classes of patterns.

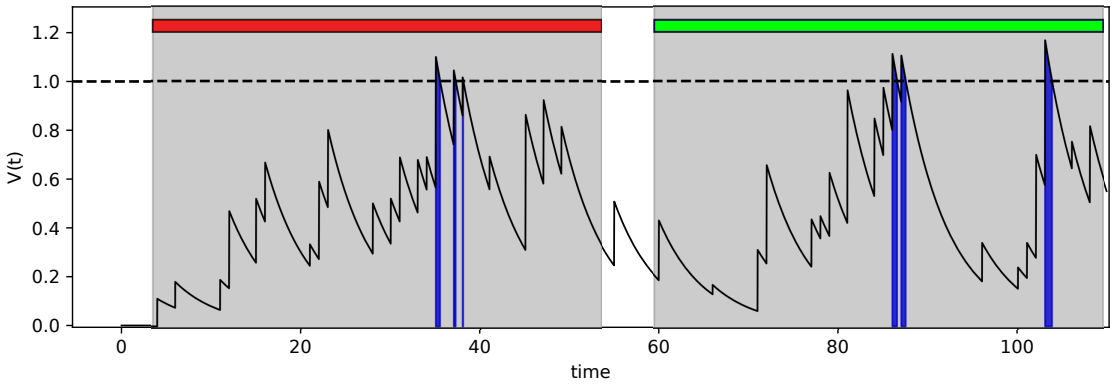


Figure 23: Pattern responses learnt in a continuous-time GNM trained by aggregate-label algorithm. The integral of super-threshold membrane potential $\mathcal{S}$ is marked in blue. In order to determine the spikes and thus the class membership, the membrane potential is integrated when it is above a readout threshold ϑ_{R}. In this particular example, the continuous spike integral totals at ~ 0.81 for the red pattern, and ~ 1.98 for the green pattern. In the case of the red pattern, the membrane potential crosses the threshold 3 times, however the total integral is ≈ 1, thus the class membership is 1 rather than 3.

3.9 Chapter summary

In this chapter, I probed the ability of the minimal neuronal model for solving multi-label classification tasks. The main contribution of this chapter is the GNM, which can be trained using aggregate-label learning originally proposed by Gütig (2016). It is demonstrated that its computing power is on par with the MST, while vastly simplifying the computation. Importantly, this model follows a "conservation of membrane potential", which prevents the arbitrary manipulation of the neuron's state. In that sense, the model is physically plausible, which allows it to be interpreted as a chemical reaction network.

While novel and powerful, Gütig's model is also internally complex, and efficient implementation outside of digital computers could pose a number of challenges. The

multi-spike tempotron can classify spatio-temporal patterns into multiple classes, however in this chapter I have shown that a much simpler neuronal model can achieve the same performance. Key components of the MST were systematically removed in order to show that they are indeed unnecessary for learning in the aggregate-label setup. The autocorrelation of the membrane potential was identified as a crucial parameter of the neuron. Moreover, this research has indicated that there exists a certain range of intermediate values for this parameter which provide best performance.

The parameters of the GNM can be readily interpreted, allowing an intuitive understanding of what enables a single neuron classification to work. The two key parameters which control the properties of the GNM are α and η. As discussed in section 3.2, parameter α describes the "memory" of the neuron. More precisely, it determines the rate of leakage for the membrane potential. At one extreme case: $\alpha = 1$, the neurons state is being reset at each time-step. The neuron has no memory of previous inputs, and thus loses the ability to integrate in time. This severely limits the GNMs ability to compute. When the memory parameter is set to $\alpha = 0$, the neuron's state has no leak, and therefore constantly integrates over its inputs. This obviously is also a suboptimal strategy, because the neuron can then only produce one spike, as its state forever remains above threshold. The optimal choice of α seems to lie somewhere in-between these two cases. A more in-depth analysis has shown that at the low end of the parameter range, there is a critical value of α which separates models with the weakest performance from the ones which learn best, see figs. 13c and 15a.

The other key parameter η can be interpreted as model choice parameter, it determines the influence of the behavioural threshold and hysteresis on the neuron's state. At one extreme $\eta = 0$, the neuron's state decays exponentially with a rate of α. Here, the neuron dynamic behaviour is not influenced by the threshold, and does not produce spiky responses. At $\eta = 1$, however, the membrane potential has no leakage whatsoever. The neuron has a perfect memory of past inputs, until it resets when the output spike is elicited. At intermediate values of η, the behavioural threshold parameter ϑ_{B}

has a greater impact on the internal dynamics of the model. When the membrane potential approaches the threshold, a decay term R rapidly increases. This results in a membrane potential decay, which increases in rate when the neuron crosses the threshold.

Further analysis has shown that the best performing models tend to lie on an off-centre diagonal in the bottom left part of the heatmaps presented in figures 13 and 14. Notably, it is uncertain whether there exists a single optimal model along this diagonal. A crucial conclusion from this analysis is that the models with $\eta = 0$, and therefore no "spikiness", perform reasonably well compared to other settings. The results from the heatmaps suggest that there may not be any benefit in strict enforcement of spiking, at least when considering this particular task. This opens up an interesting discussion, whether the hysteretic behaviour as defined in this chapter is necessary for efficient learning.

These findings, also point to a different insight, the temporal autocorrelation of the membrane potential emerges as the crucial parameter of this model. It is important to point out, that the optimal diagonal, remains a line of "constant memory". This is a result of the factor $(1 - \eta)$ in the membrane potential equation (see eq. 31), which additionally reduces the degree of the memory parameter α. This observation suggests that, while there is no strict optimal "spikiness", there indeed is an optimal value for "memory", at least in the context of patterns with the length and firing statistics considered in this research. Nevertheless, the GNM does not seem to be particularly sensitive to small changes in α, and there exists a wide range of parameters which provide for a decent performance.

The simplified version of the GNM with $\eta = 0$ reduces to a spike integrator with a fixed leakage rate, and lends itself to an interpretation as a chemical system. The function of V can be interpreted as a concentration of certain molecular species with a fixed decay constant: $V \xrightarrow{\alpha} \varnothing$. This trend is only interrupted by immediate increases of the membrane potential, caused by the pre-synaptic inputs: $V \longrightarrow kV$. Moreover, the

synaptic weights associated with different input channels can be implemented through modifying certain reaction rate constants. Each of N input channels I_i decays to V with a rate of $w_i C$ and to $\varnothing$ (outside of the system) with a rate of $(1 - w_i)C$. This finding brings an interesting perspective on how a chemical system can implement signal modulation. Nevertheless, this method still requires an external observer to apply the algorithm, and calculate new w_i at each post-synaptic spike. Moreover, the storage of information about the current weights is not internal to the system, and adjusting rate constants may not be as trivial in a real biochemical system. While training could pose a challenge, performing classification tasks with an already trained model with fixed rate constants could be possible. The only feature which both the MST and this chemical system share is the ability to integrate and remember past inputs between time steps. This again leads to the conclusion that the "memory" parameter is crucial for multi-label classification.

The GNM's performance was compared to other neuronal models, such as LIF and MST, trained with the aggregate-label learning algorithm. The LIF neurons typically exhibit a hard reset after an output spike is elicited. The membrane potential subsequently enters the refractory period, when the neuron is incapable of processing more incoming signals for a fixed period of time. This feature has previously been found in real neuronal systems (Feldman 2012). It is also useful in layered approaches, in particular in combination with STDP (Gerstner and Kistler 2002a), where it implements a form of *winner takes all* dynamics. This prevents all neurons in the network from reacting to the same stimulus, thus allowing for more refined classification. Otherwise, the advantages of using the refractory period in single artificial spiking neurons is unclear. It is important to emphasise, that when post-spike reset and refractory period are removed, the LIF neuron becomes mathematically equivalent to the GNM with $\eta = 0$.

The simulations have shown that while the performance of the LIF neuron is on par with both the GNM and the MST in the case of a single pattern, it quickly drops as soon as multi-class learning is introduced. This seems to point to the conclusion that

the refractory period harms the performance in the latter case. This stems from the fact that the refractoriness affects the period of time when the neuron can react to incoming spikes. In the multi-spike framework, this severely impacts the performance as the LIF neuron is incapable of releasing several spikes within a limited time window when the pattern is presented.

Nevertheless, the performance of the LIF neuron is worse than that of the GNM even for a refractory period of length 0, see fig. 16. As pointed out earlier, in this case the only difference between the LIF and the GNM is the immediate post-spike reset of the membrane potential. This has an effect of immediate forgetting of the past inputs, thus no temporal autocorrelation of the inputs is preserved after a single output spike is produced. This begs the question whether biological neurons, which do have a refractory time, are sub-optimal components. This conclusion is far-fetched, and most likely invalid, since the real neuronal systems operate in a different context than the limited problem examined here. Furthermore, the refractory duration in real neurons could be a reflection of resource restrictions, or otherwise some physical constraints that were not considered here.

The majority of tasks considered in this chapter were trained using aggregate-label learning rule, see section 3.3. Here, the feedback is provided with a delay at the end of each training phase. An aggregate error value is calculated and distributed among the weights based on their spiking history. Gütig (2016) has motivated this method by its presumed biological realism, in particular NMDA receptors could in an abstract way facilitate a similar process. While aggregate-label learning remains an interesting learning rule, and the biological plausibility was the key criterion in designing it, it is not necessarily the most effective for this tasks. Firstly, a more information-rich error-feedback method has been proposed. Dropping the requirement of delay and aggregate-label for the error trace resulted in a substantial increase in the model's performance, see section 3.6. This direct error feedback also allows for backpropagation-based training in layered networks of GNMs, see fig. 18. The feasibility of this method was

demonstrated by solving a multi-label classification task using a hierarchical network of GNM with 10 hidden neurons.

Chapter 4

Chemical neuron and computation in cells

4.1 Introduction

Intelligence is typically associated with animals with nervous system (Walters, Carew and Kandel 1979; Fanselow and Poulos 2005). However, a number of other plausible substrates capable of simulating intelligent behaviour have been proposed (Adamatzky et al. 2019), one of them being biochemical systems (Amos 2004). In nature, even single-celled organisms show the ability to analyse and act upon changes in their environment. These intelligent patterns of behaviour are implemented through bio-molecular circuits in their entirety. Perhaps the most well-documented example of such intelligent behaviour is chemotaxis, which enables $E.coli$ bacteria to readily adapt to changes in concentration of certain chemicals in their environment (Yi et al. 2000; Hoffer et al. 2001). Other examples of biochemical information processing is sensing (Govern and ten Wolde 2014a,b; Alon 2019) or diauxic growth (Chu 2018, 2017; Chu and Barnes 2016).

The advancements in synthetic biology make it possible to implement increasingly

more complex systems in wet laboratories. Artificial devices capable of intelligent behaviour could be useful in a vast array of use-cases. Examples include environmental sensing and cleaning (Schneiker et al. 2006), smart drug delivery (Ausländer, Wieland and Fussenegger 2012), or to enable more precise control in bio-reactors for the production of drugs (Trosset and Carbonell 2015). Designing such systems has proven to be challenging due to a range of requirements, such as the need for the system to be low-powered, small, and autonomous.

Perhaps the most important requirement for implementing neuronal systems in the context of synthetic biology is their *autonomy*. The neurons learn by adjusting the strengths of their synaptic connections. In digital computers this can be easily done, however in chemical computers this becomes more difficult due to the information being encoded in terms of reaction rates and molecular abundances. The spiking neuron models discussed in the previous sections only describe the integration of pre-synaptic signals and implement a threshold for output spiking. These models rely on an external learning algorithm, such as STDP, to be applied under certain conditions. Moreover, such learning assumes a memory of pre-synaptic inputs, in order to adjust the weights.

Therefore, I propose that the following list of requirements needs to be met in order to consider a system fully autonomous in the context of biochemical implementation:

1. The systems needs to be able to learn in an unsupervised way, since providing feedback would require an external agent.

2. The learning algorithm needs to be implemented within the reaction system itself. This means that the simulation cannot be stopped to perform weight adjustment.

3. The storage of synaptic weights cannot rely on external storage. They need to be represented internally by the system.

4. Signal modulation also needs to be internal to the system. There needs to be a mechanism which allows the weights to scale the impact of their respective inputs on the state of the neuron.

A number of attempts have been proposed to designing chemical implementations of intelligent systems. The earliest attempts to implement intelligent systems as biochemical circuits focused on artificial neurons. Examples include work by Okamoto, Sakai and Hayashi (1988) or Hjelmfelt, Weinberger and Ross (1991) who designed biochemical network capable of simulating a McCulloch-Pitts neuron and act as logic gates. A biochemical perceptron model was proposed by Banda, Teuscher and Stefanovic (2014). This was later used to build feed-forward networks of perceptrons capable of solving the XOR problem (Blount et al. 2017). Nevertheless, neither of these approaches was capable of learning, as they only presented models for neural integration of rate-coded inputs. More recently a number of new substrates for computation in biology have been identified, for example: interconnected phosphorylation/dephosphorylation which can implement biomolecular neural networks (Samaniego et al. 2020; Moorman et al. 2019), microbial consortia capable of simulating perceptron neurons (Li et al. 2021), and receptor-ligand interactions in the bone morphogenetic protein (BMP) pathways used to simulate simple neuronal functions (Antebi et al. 2017). Besides neuronal circuits other researchers focused on more fundamental tasks such as associative learning, for example in gene regulatory networks (Fernando et al. 2009) or in multi-cell systems (Macia, Vidiella and Solé 2017a; Macia and Sole 2014; Macia, Vidiella and Solé 2017b). Other researchers also attempted to implement multiple forms of *in vivo* computation (Shirakawa and Sato 2013; Nesbeth et al. 2016; Chen and Xu 2015; Racovita and Jaramillo 2020).

Fulfilling the requirements for autonomy of the system proves to be a difficult task. None of the aforementioned models met all of the criteria established earlier. Some of them only present classification models, others require an intervention from an external observer during the training phase, or can only learn a single association and thus become useless for solving more complex spatio-temporal tasks.

In this chapter, I will probe the question of the simplest autonomous learning system implementable as a biochemical circuit. I will propose the chemical neuron (CN)

model constructed as a molecular system following mass-action kinetics. The model is described as a chemical reaction network, and mimics the behaviour of a spiking neuron. The results part of this chapter will commence by demonstrating that the system is capable of associative learning. Next, I will demonstrate that the CN can implement a full Hebbian learning mechanism, in the sense that its internal molecular abundances will encode statistical biases of its inputs. More specifically, two types of bias will be considered: frequency bias (FB) and temporal correlation (TC). In the former, the spiking frequency of certain synapses is increased. Thus, the input times are drawn from independent, but differently distributed random variables. In the latter, the input frequency remains the same across channels, however some of the inputs always spike in close temporal proximity. Thus, the input signals are drawn from identically distributed, but not independent random variables.

Through extensive simulation, it will be demonstrated that the CN can learn both of these tasks. Learning capabilities of the system will be examined and I will attempt to determine the optimal chemical composition for detecting different types of bias in the data. In particular the parameter which controls the degree of nonlinearity of the CN's *activation function* will be considered. In order to estimate the thermodynamical cost of computation I will measure the entropy production of the system as a function of its volume and other key parameters. The tradeoff between the cost of computation and quality of the weights will also be examined. While thermodynamically consistent, the CN model cannot be easily realised in synthetic biology. To this end, I will propose an interpretation of this model as a single-celled organism, built using well-known biochemical motifs. While not thermodynamically explicit, this version is however biochemically plausible and could potentially be synthesised in a laboratory. Again, I will demonstrate that it's capable of both associative learning and implements full Hebbian learning extendable to an arbitrary number of input channels.

4.2 Frequency accumulator model

I will first consider a model capable of learning to recognise bias in the spiking frequencies of multi-channel inputs. This task reflects the limited case of learning in rate-coded neurons, where the temporal aspect of the inputs is ignored. Perhaps the simplest computational unit capable of that can be implemented by a chemical reaction network with just five reactions. Figure 24 shows the schematic depiction of such a system.

Here, the species $A_1, \ldots, A_N$ are the *input species* of the CN. They can be used to encode some external information to be processed by the CN. This means that at various time points $t_n^s \in \mathbb{R}^+$ a number of $M > 0$ molecules of I_n are added, this value will be henceforth referred to as the "bolus". Therefore adding the molecules is not modelled as an instantaneous process. Instead, it can be thought of as that at each of the time-point t_n^s the CN is brought in contact with a reservoir consisting of M precursor molecules I_n that then decay into A_n molecules with a rate constant $\kappa > 0$.

Once inside the system, the input molecules decay to a molecular species B with a rate constant k_{AB}. In neuromorphic analogy, molecule B plays a role of the *membrane potential* encoding the internal state of the system. Its function is to integrate the incoming signals over time, and thus it serves as a form of short-term memory of the neuron. This kind of trivial system can detect frequency bias in the input data. This process is facilitated by a less reactive molecule H_n which could be thought of as the *weight* of the input channel n. This type of long-term memory is formed by a slow reaction $\mathrm{A}_n \overset{\mathrm{k_{AH}}}{\rightleftharpoons} \mathrm{H}_n$. Therefore, the input channels which "spiked" more frequently have more H_n associated with them. Notably, such a system would not be able to implement learning of temporal correlations, as it lacks a time-dependent activation function.

Function	Reaction
Input	$I_n \underset{k_{AI}}{\overset{k_{IA}}{\rightleftharpoons}} A_n$
	$A_n \underset{k_{BA}}{\overset{k_{AB}}{\rightleftharpoons}} B$
Weight accumulation	$A_n \underset{k_{HA}}{\overset{k_{AH}}{\rightleftharpoons}} H_n$
Leak	$H_n \underset{}{\overset{k_{HOut}}{\rightleftharpoons}} Out$
	$B \underset{}{\overset{k_{BOut}}{\rightleftharpoons}} Out$

Table 1: List of chemical reactions in a single frequency accumulator CN unit.

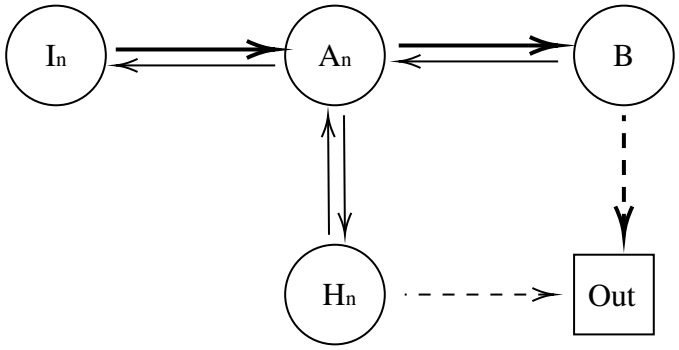


Figure 24: Schematic representation of a basic chemical reaction network which can accumulate information about input frequencies of different synaptic channels.

4.3 Chemical neuron model

The main contribution of this chapter is the model of the chemical neuron (CN). CN is fundamentally constructed as a set of micro-reversible chemical reactions. Each reaction is formulated exclusively in terms of mass-action dynamics. The law of mass-action describes how the velocity of a chemical reaction is related to the molecular concentrations of the reactants, and thus underpins certain biochemical phenomena, such as molecules binding to receptors. The full CN model is an extension of the frequency accumulator model, which additionally includes reactions responsible for an activation

Function	Reaction
Input	$\mathrm{I}_n \underset{\mathrm{k_{AI}}}{\overset{\mathrm{k_{IA}}}{\rightleftharpoons}} \mathrm{A}_n$
	$\mathrm{A}_n \underset{\mathrm{k_{BA}}}{\overset{\mathrm{k_{AB}}}{\rightleftharpoons}} \mathrm{B}$
Activation function	$\mathrm{B} + \mathrm{E}_i \underset{\mathrm{k^-}}{\overset{\mathrm{k^+}}{\rightleftharpoons}} \mathrm{E}_{i+1}, \qquad i < m-1$
	$\mathrm{B} + \mathrm{E}_{m-1} \underset{\mathrm{k^-_{last}}}{\overset{\mathrm{k^+}}{\rightleftharpoons}} \mathcal{E}$
Weight accumulation	$\mathrm{A}_n + \mathcal{E} \underset{\mathrm{k_{EA}}}{\overset{\mathrm{k_{AE}}}{\rightleftharpoons}} A\mathcal{E}_n \underset{\mathrm{k_{HE}}}{\overset{\mathrm{k_{EH}}}{\rightleftharpoons}} \mathrm{H}_n + \mathcal{E}$
Signal modulation	$\mathrm{A}_n + \mathrm{H}_n \underset{\mathrm{k_{HA}}}{\overset{\mathrm{k_{AH}}}{\rightleftharpoons}} \mathrm{AH}_n \underset{\mathrm{k_{BH}}}{\overset{\mathrm{k_{HB}}}{\rightleftharpoons}} \mathrm{B} + \mathrm{H}_n$
Leak	$\mathrm{H}_n \underset{}{\overset{\mathrm{k_{HOut}}}{\rightleftharpoons}} \mathrm{Out}$
	$\mathrm{B} \underset{}{\overset{\mathrm{k_{BOut}}}{\rightleftharpoons}} \mathrm{Out}$

Table 2: List of chemical reactions in a single chemical neuron.

function, signal modulation, and threshold mediated weight accumulation. These differences allow the CN model to learn temporal correlations in the multi-channel input stream. The reaction are tabulated in table 2, for schematic diagram see fig. 25, and for the reaction rate constants see table 3.

The system can be best understood by first considering the species which have their equivalent in other spiking neuron models. Similarly to the frequency accumulator model, the molecular species A_i can be thought of as the synaptic input to the system via input channel i. The internal state or the membrane potential of the CN is represented by the abundance of the B molecules. The abundances of the species H_i indicates the weight associated with the i-th input channel of the CN. Lastly, the molecular species $\mathcal{E}$ which is the activated form of E has the role of the learning signal and the output of the neuron. There are four main reactions which facilitate learning in such a system, I will now discuss them in more detail starting with the input integration.

Function	Reaction rates
Input	$k_{IA} = 10$, $k_{AI} = 0.000001$
	$k_{AB} = 0.1$, $k_{BA} = 0.000001$
Activation function	$k^{+} = 1$, $k^{-} = 5$
	$k^{-}_{last} = 0.5$
Weight accumulation	$k_{AE} = 0.05$, $k_{EA} = 0.000001$, $k_{EH} = 100$, $k_{HE} = 0.000001$
Signal modulation	$k_{AH} = 0.001$, $k_{HA} = 0.000001$, $k_{HB} = 100$, $k_{BH} = 0.000001$
Leak	$k_{H\varnothing} = 0.0003$
	$k_{B\varnothing} = 0.1$

Table 3: List of reaction rate constants in a single chemical neuron.

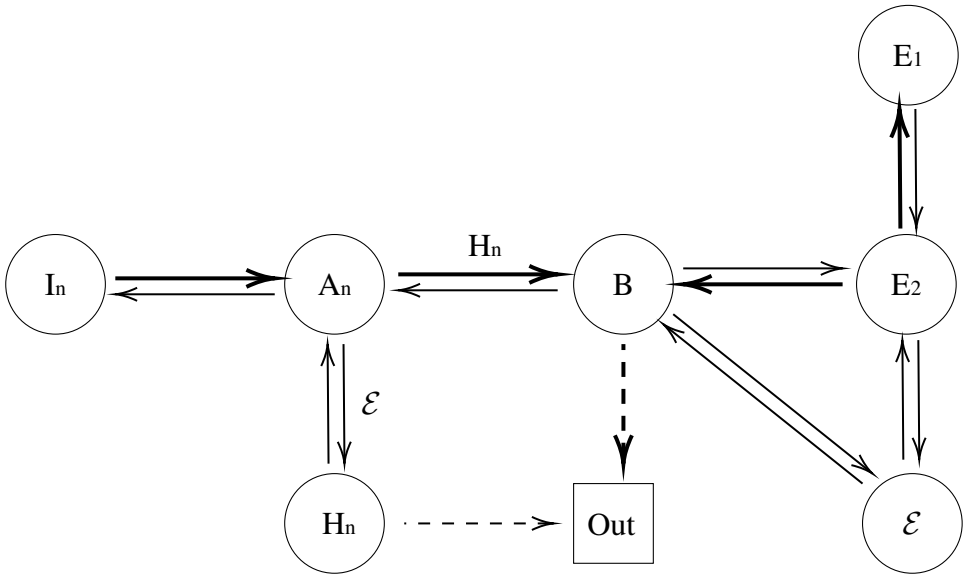


Figure 25: Schematic representation of a single chemical neuron (CN) processing unit as chemical reaction network. The bold arrows signify reactions where one of the rate constants is significantly stronger than its reverse Annotated arrows depict catalytic reactions.

Input integration

The input integration is implemented in the same way as proposed in section 4.2. Therefore, the species $A_1, \ldots, A_N$ are the *input species* of the CN. Again at each of the timepoint t_n^s a predefined amount of precursor molecules I_n are injected. The amount of I_n added to the system will be henceforth referred to as *bolus size* or β. The I_n molecules, in turn, decay to their respective A_n with a rate constant $\kappa > 0$.

Once inside the system the input molecules partake in a number of reactions. The first one is that they decay themselves to a molecular species B with rate constant k_{AB}. The molecule B plays a central role in the system in that it encodes the internal state of the CN. In the spiking neuron analogy, one could think of B as the *membrane potential*. Its function is to integrate the incoming signals over time, thus it serves as a short-term "memory" of the system.

Activation function

The next functional module of the CN is the activation function, see table 2. This module involves the molecules E, which simulate a ligand receptor with m binding sites that can be occupied by molecules of type B. Association and dissociation of B molecules happens with fixed rates. *Cooperativity* is a phenomenon that leads to collective properties of chemical systems, which are not present on the level of individual molecules. More precisely, it describes a behaviour where binding of a ligand to one site influences the rate of subsequent bindings. This model implements a form of cooperativity in that the binding affinity is much lower when all binding sites of E are occupied. This is implemented by setting the dissociation constants to $k^-_{\mathrm{last}} \ll k^-$.

With an appropriate choice of rate constants, this system is known to implement ultrasensitivity, i.e. the probability for the fully occupied form of the ligand chain ($\mathcal{E}$) to exist transitions rapidly from close to 0 to close to 1 as the concentration of ligands approaches a threshold value $\vartheta \approx k_+/k_-$ (Chu, Zabet and Mitavskiy 2009). Such

systems can be approximated by the Hill kinetics:

$$f(x) = \frac{x^h}{\vartheta^h + x^h} \tag{38}$$

where h is the parameter which determines the steepness of the function, and ϑ denotes the transition point or the activation threshold. It can be shown that the maximal Hill exponent that can be achieved by such a system is m. Therefore, the number of binding sites on the receptor determines how similar the activation function is to a step function. The parameter m will be from now on referred to as the *chain length* or *nonlinearity*. The degree of nonlinearity in the activation function will turn out to be a crucial parameter determining the computational properties of the CN.

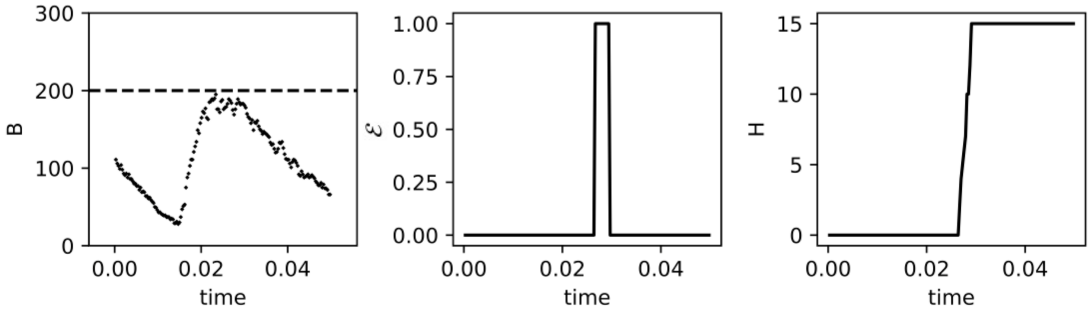


Figure 26: The first graph shows the evolution of membrane potential (B) over time. The membrane potential increases in response to external input presented to the CN at time 0.015. Secondly, I examine the behaviour of $\mathcal{E}$ molecules representing the last link in the ligand chain, i.e. an activate form of the molecule (for $m = 15$). As membrane potential approaches the threshold, more ligands bind to the receptors and eventually $\mathcal{E}$ molecules are produced. When the amount of B in the system drops due to B $\overset{\text{k}_{\text{BOut}}}{\rightleftharpoons}$ Out reaction, the ligands unbind and $\mathcal{E}$ molecules dissipate again. Lastly, the third graph shows how the amount of H molecules changes over time. Here, since the $\mathcal{E}$ molecules become present approximately at time 0.03, the reaction $\text{A}_n \overset{\mathcal{E}}{\rightleftharpoons} \text{H}_n$ is catalysed and the H molecules are accumulated. Note, that the CN reinforces the weight associated with the input which triggered the threshold crossing event. The temporal coincidence of A_n and $\mathcal{E}$ molecules acts as a Hebbian element which drives the increase of the weights.

Weight accumulation

Associated with each input channel n is a slowly acting learning module and its associated molecule H_n, which can be thought of as the *weight* of the input channel n. In the next sections, I will demonstrate that the learning reaction the CN implements is akin to the Hebbian learning rule. Fundamentally, it implements a slowly acting feedback loop in that a higher concentration of A_n accelerates the accumulation of the weight molecule H_n, which in turn accelerates the conversion of input A_n to the state molecules B. Note that this learning module is only active when the $\mathcal{E}$ molecules are present, which in turn only happens when the state molecules B are above a threshold abundance. Finally, the CN also has the ability to forget its past history, which is implemented by the decay of the weight and state molecules outside of the system.

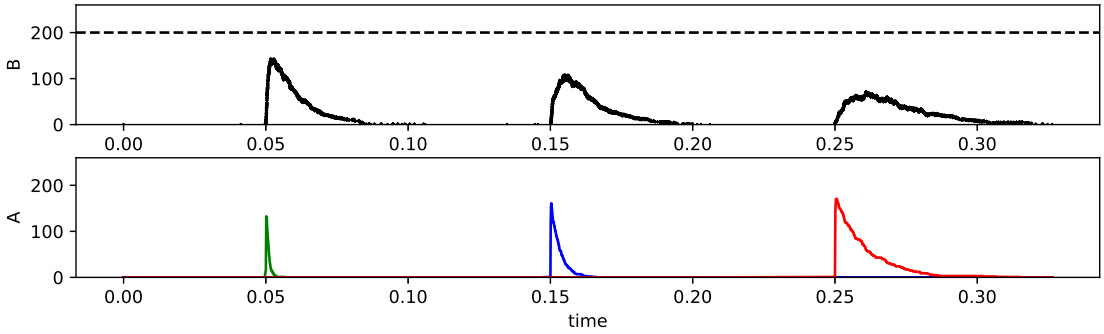


Figure 27: Example of three inputs of uniform size received from 3 different channels. Each input shown in the second graph has a different weight associated to it: $H_{\text{green}} = 250$, $H_{\text{blue}} = 50$, and $H_{\text{red}} = 0$. H molecules act as a catalyst in $\mathrm{A}_n \overset{\mathrm{H}_n}{\rightleftharpoons} \mathrm{B}$ reaction, hence the change in the function of B molecules over time for each of the inputs. The higher the amount of H, the higher is the peak of B molecules caused by a particular input. Moreover, with the increase in weights, the function of inputs also changes. The higher the amount of H, the quicker its corresponding A dissipates. Therefore, the chance to further increase this weight in the future updates decreases, which introduces a self-regulatory mechanism similar to Oja's rule (Oja 1982) (see fig. 28).

Signal modulation

Figure 27 shows the effect of different weights on the function of inputs and the membrane potential. This reaction takes a catalysed, as well as an uncatalysed form. The

uncatalysed reaction $A_n \underset{k_{BA}}{\overset{k_{AB}}{\rightleftharpoons}} B$ is necessary in order to allow the system to learn to react in response to new stimulus, even when the weight associated with a given channel decayed to 0. In the case of the catalysed reaction the channel specific H_i molecules play the role of the catalyst. Thus the amount of H_n molecules controls the conversion speed of input A_n to the membrane potential B. This in turn changes the influence of each input channel on the state of the neuron. This mechanism, however, also implements a self-regulating mechanism which prevents the weights from growing indefinitely, see fig. 28.

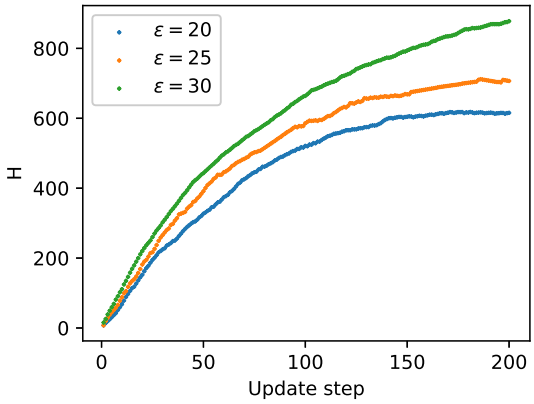


Figure 28: Self-regulation mechanism in the CN is similar to the one proposed by Oja (1982). This graph shows the rate of change of a weight as a function of the update step number. As the weight increases the subsequent update becomes smaller, as an effect of faster dissipation of A. Eventually, the weight stabilises at the equilibrium level, which is determined by parameter ϵ - the total amount of E_1 molecules in the system at the beginning of the simulation.

To summarise, the mechanisms implemented by the CN can be described in a simplified way:

1. Input signals are added to the system by molecular species $A_1, \ldots, A_N$ representing the input channels.

2. The inputs are integrated and converted into a molecular species representing the membrane potential of the neuron: B.

3. The weight of each input channel is represented by molecular species H_n. These weight molecules decay outside of the system via reaction $\mathrm{H}_n \xrightleftharpoons{\mathrm{k_{HOut}}} \mathrm{Out}$.

4. If the amount of the membrane potential species reaches a threshold, then channels accumulate weights via molecular species H_n. How much weight they accumulate depends on the abundance of the input molecules A_n. Those channels that have received input immediately before the system reached the threshold will have more input molecules. This implements the Hebbian idea of "what fires together, wires together."

5. More weight molecules means that future inputs are converted faster to the state molecules. Therefore, they have a greater immediate effect on the post-synaptic membrane potential.

6. However, larger weights also mean that the inputs dissipate faster, thus making it more difficult to accumulate more weights in the future. In other words, effectively decreasing the learning rate, in a way similar to Oja's rule (Oja 1982).

4.4 Learning

The experiments conducted in the following sections were simulated using Gillespie's stochastic simulation algorithm (Gillespie 1976) in purpose-built software written in Python [1]. The software generates a statistically correct trajectory of the chemical reaction network by simulating collisions of molecules within an enclosed environment. The reactions in the Gillespie's algorithm must involve at most two molecules, and the reaction environment is assumed to be well mixed.

[1] The details of the software implementation are available on https://github.com/jf330/CRN-neuron/

4.4.1 Associative learning

Firstly, I will demonstrate that the CN can solve the task of associative learning with $N = 2$ input channels only, see fig. 29. This corresponds to a well-known case of Pavlovian conditioning. Previous molecular implementations of associative learning, for example by Fernando et al. (2009) (see section 2.6.2), did not offer much flexibility and could only learn a single pre-defined set of inputs. The system I propose can learn arbitrary sequences and requires several coincidences before it learns the association, and thus is robust against noise. It also can unlearn the correlation if input patterns change.

The model is initialised in such a way that the weights associated with the first channel are high ($H_1 = 100$) and the weights for the second channel are low ($H_2 = 0$). With this initialisation setting a single bolus of A_1 is sufficient to push the state molecule over the threshold of $\vartheta = 250$ molecules of B. In contrast, a bolus of A_2, corresponding to stimulating the second channel is not sufficient to trigger a response. Crucially, a bolus of A_2 will not lead to an increase of the weight H_2 associated with the second channel. If, however, both A_1 and A_2 are given simultaneously, then the action of A_1 is sufficient to push the internal state over the threshold; at the same time temporal proximity of the bolus with the threshold crossing leads to an increase of the weights H_2 of the second channel. After a few of coincidence of A_1 and A_2, the weight associated with the second channel reaches a sufficient value and A_2 can push the internal state of the system over the threshold on its own.

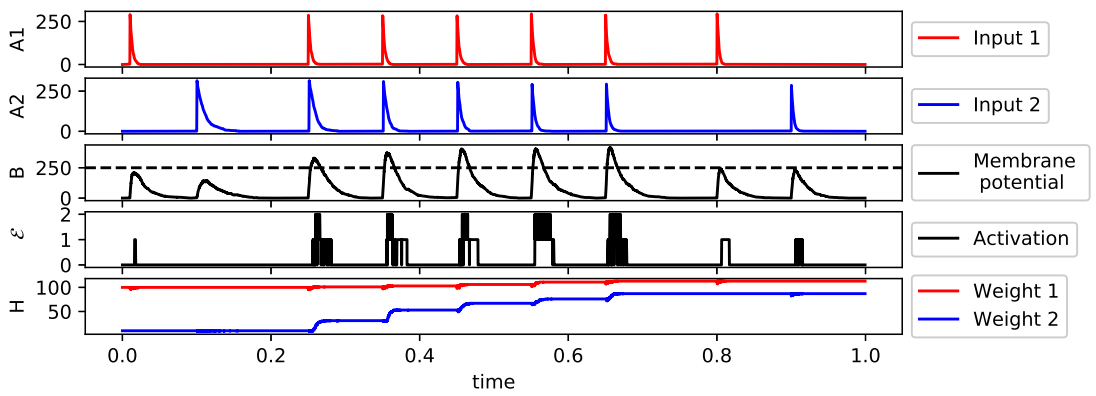
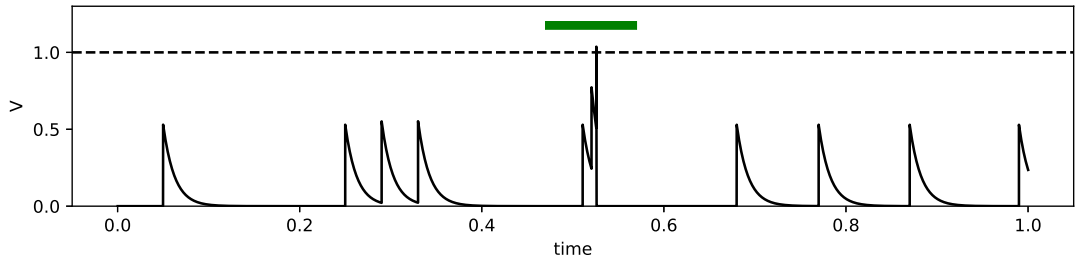


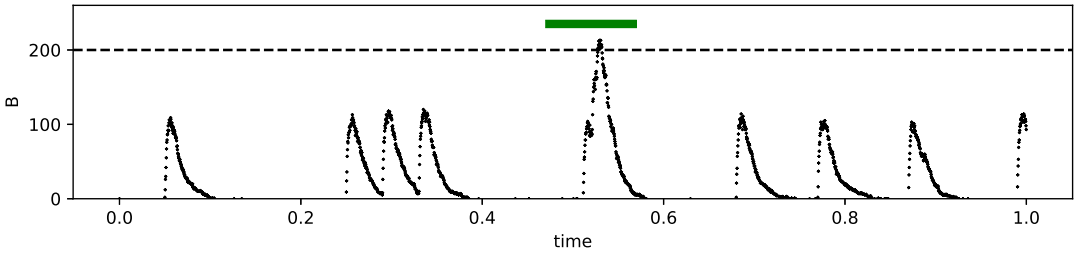
Figure 29: Example of associative learning in the CN. The CN is presented with two inputs: A_1 and A_2. At first, $\mathcal{E}$ molecules are only produced in response to the first input, and none are produced when a signal from the second one is presented. Next, the CN is exposed to both inputs in close temporal proximity multiple times. The fourth graph shows that the weight associated with the second input channel H_2 grows each time the inputs coincide, however H_2 is saturated at this point. Lastly, at time $t = 0.9$ the neuron is presented with the input A_2 only. After just a few coincidences the neuron produces an output spike in response to the input from A_2 without any additional stimulation.

4.4.2 Full Hebbian learning

Next, I will test the CN on the task typically associated with computation in spiking neurons (Gütig and Sompolinsky 2006; Brunel and van Rossum 2007). The system is presented with a random stream of inputs embedded with patterns of fixed temporally correlated spikes from the subset of pre-synaptic channels. Figure 30 shows the behaviour of SN and equivalent CN trained on statistically the same randomly generated input. In this example, the task is to recognise a single spatio-temporal pattern consisting of 3 spikes, where the input from I_1 was followed by the one from I_0, and another input from I_1. It is worth noting that the SNs are typically trained on data which contains both temporal and statistical bias.



(a) LIF neuron



(b) Chemical neuron

Figure 30: State of a (a) LiF neuron, and (b) equivalent CN given the same input. At time 0.5 a trained pattern is presented. This causes the activation of both SN and CN to exceed the threshold. The discrete nature of absolute refractory period in the SN, which causes the membrane potential to immediately drop to 0, cannot be reproduced in this chemical network.

In the following sections, however, I will focus on showing that the CN is capable of learning to recognise both the frequency bias as well as temporal correlations embedded within its input streams, in the sense that its internal molecular abundances will reflect these statistical biases.

Learning frequency bias

First, I will test the CN on the task which requires it to recognize frequency bias in the input data, the same as in section 4.2. This type of task will be henceforth referred to as the FB task. In practice, this is done by providing boli to each of the N input channels at random times. This means that the waiting time between two successive boli of A_i is distributed according to an exponential distribution with parameter $1/f_i$, where f_i is the frequency of the input boli to channel i. The CN should then detect the

difference in frequencies f_i between input channels. At the steady state, the ordering of the abundances of weights reflects the input frequencies, i.e. the number of H_i should be higher than the number of H_j if $f_i > f_j$.

The CN is initialised with $N = 5$ input channels and the weight molecules are set to $H_i = 0$ at the beginning of each simulation. In order to test this, 3 variants of the FB task are considered. The variant FB 2 assumes that boli to the first two input channels come at a frequency of 4Hz whereas channels 3, 4 and 5 fire at a frequency of 2 Hz; Similarly, for tasks FB 3 and FB 4 the first 3 and 4 channels respectively also spike at the high frequency. Fig. 31 shows the steady state weights for each of the three tasks. As expected, in each of the experiments the weights of the high-frequency inputs are higher when compared to the low frequency inputs. Therefore, the CN can be used as a frequency detector.

Learning temporal correlations

In this section, I will demonstrate that the CN is capable of detecting temporal correlations in the input data stream. I will henceforth refer to this task as a TC task. In this scenario, all inputs are added to the system with the same frequency, i.e. $f_i = f_j$ for all pairs of channels i and j. For some pairs of channels, the probability to observe an input bolus A_k within a time period τ after a bolus A_l is higher than within a time period τ before a bolus of A_l. In practice, such correlations are implemented as follows: If A_1 and A_2 are temporally correlated then each bolus of A_1 is followed by a bolus of A_2 after a time period of δ. In all simulations presented below, the input frequency of all channels is set to 2Hz. After a transient period, the weights should indicate which channels are correlated and the temporal order implied by the correlation, i.e. if A_i tends to precede A_j, then the corresponding weights should be $H_i > H_j$.

In order to test the ability of the system to detect TC biases, the CN is initialised with $N = 5$ input channels and the weight molecules are set to $H_i = 0$. Similarly to the FB case, three different scenarios are compared where there are correlations between

A_1 and A_2 (TC 2), A_1, A_2, A_3 (TC 3), and A_1, A_2, A_3, A_4 (TC 4). The temporal order is always in ascending order of the index, such that in the last example, A_1 occurs before A_2, which in turn occurs before A_3. Fig. 31 shows that at the steady states, the accumulated weights reflect the correlation between input channels, including the temporal ordering.

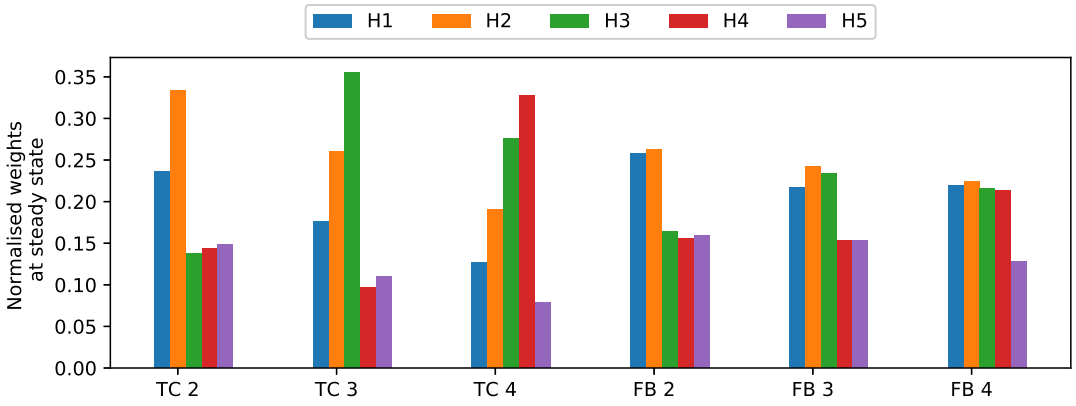


Figure 31: Normalised weights for a variety of TC and FB tasks. Here, each bar corresponds to a different weight, where the blue bar refers to the first weight, orange to the second etc. The nonlinearity was set to $m = 4$ for the TC, and $m = 1$ for FB, and the initial number of E_1 molecules was set to 40. Each values were obtained by averaging the weights at the steady state over 300 time units, after the neuron was trained for 700 time units. The CN is trained for 3 tasks with temporal correlation of inputs, where 2, 3 or 4 synapses are biased respectively. Similarly for FB, where 2, 3 or 4 synapses have the higher spiking frequency. The weight distributions at the steady state proves to be highly correlated with the type of bias embedded in the data.

4.5 Performance analysis

4.5.1 Nonlinearity and learning

I will now examine in more depth the impact of the nonlinearity parameter m on the ability of the system to compute. As noted before, m represents the number of receptors on the ligand chain, and thus governs the steepness of the activation function. Let's consider two extreme cases: the case of minimal nonlinearity (i.e. $m = 1$) and the

case of maximal nonlinearity (i.e. $m = \infty$). Where the latter case corresponds to an activation function similar to a step function. To be precise, biochemical systems such as the CN cannot realise a pure step function, nevertheless this abstraction provides a useful insight into the properties of the system.

I will first consider the case of maximal nonlinearity (i.e. $m = \infty$) and a CN with two input channels A_1 and A_2. In this case, there will be a learning signal $\mathcal{E}$ in the CN if the abundance of B crosses the threshold ϑ. Assume that the parameters are set such that a single bolus of either A_1 or A_2 is not sufficient to push the abundance of B over the threshold, but a coincidence of both is. Therefore, a single bolus of A_1 will not lead to a threshold crossing and the weights will not increase. However, If a bolus of A_1 coincides with a bolus of A_2 then the membrane potential will cross the threshold. As a result, a learning signal will be generated and weights for both input channels 1 and 2 will be increased (although typically not by equal amounts).

In the opposite case, i.e. $m = 1$, the neuron's activation function is minimally nonlinear. Similarly, both A_1 and A_2 are required to push the abundance of B across the threshold. However, now the learning behaviour of the CN will be different. A single bolus of A_1 will not lead to a threshold crossing, however a learning signal may still be generated even below the threshold because the activation function is a gradual change rather than a step function. As a result, weight H_1 will be increased by an amount, depending on the bolus size. If a bolus of A_1 coincides with a bolus of A_2 even more learning signal molecules will be produced than in the case of a single input. Thus, the weights for both input channels will be potentiated more than if they had occurred separately.

These two extreme cases illustrate how the CN activates in response to inputs. In the case of low nonlinearity the weights of a channel will be a weighted sum over all input events of this channel. The weights will be higher for channels whose boli arrive with a higher frequency. On the other hand, a step-like activation function will integrate only over those events where the threshold was crossed. Thus the neuron is tuned to

detect temporal coincidences. Two conjectures can be derived from these observations:

- As the degree of nonlinearity increases, the CN becomes better at detecting temporal coincidences. Lowest nonlinearity still allows for coincidence detection, however in a significantly weaker form.

- The higher the bolus size, the worse the CN's ability to detect temporal correlations. Particularly, when the bolus size is sufficiently large for a single bolus to push the abundance of B over the threshold. In this case, a single input spike would be able to saturate the activation function, thus undermining the ability of the system to detect coincidences effectively.

In an effort to validate these conjectures, consider an example of a CN with 3 inputs. Here, the inputs A_1 and A_2 are correlated and A_3 comes at a frequency twice as high as that of A_1 and A_2. Figure 32 shows the weights as a function of the bolus size. Additionally, the degree of nonlinearity was varied from $m = 1$ (minimally nonlinear case) up to $m = 4$ (moderate nonlinearity).

The CN with a minimal nonlinearity seems to detect both coincidences and frequency differences. However, the weight associated with A_3, is consistently higher than the other weights. Noticeably, it also loses its ability to detect coincidences for larger bolus sizes, as both temporarily correlated weights converge to the same value. These observations are consistent with the conjectures listed earlier.

In the case of a more nonlinear CN ($m = 4$) with moderately low bolus size the weight distribution indicates the temporal coincidence more strongly, i.e. the weights associated with the temporal coincidence are higher relative to H_3. Nevertheless, the CN still loses its ability to detect coincidences as the bolus size increases, due to a single input being able to trigger enough activation.

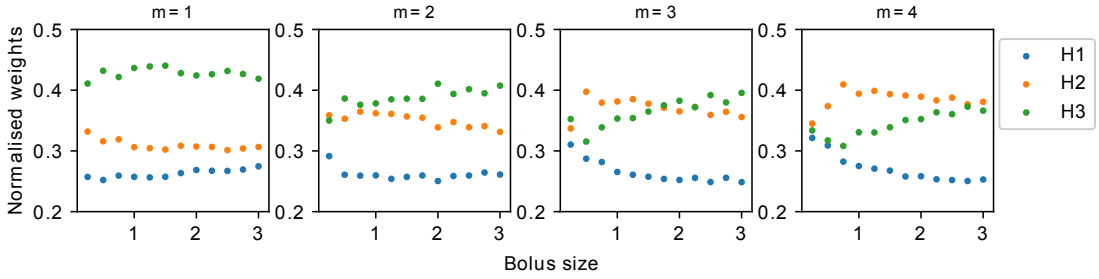


Figure 32: The weights as a function of bolus size for a CN with 3 inputs. Input A_1 (green) is provided at 4Hz, A_2 and A_3 are correlated with a $\delta = 0.0047$ but they are presented with a lower frequency of 2Hz. The graph shows the normalised weights at steady state corresponding to the input channels for different bolus size relative to the threshold. The nonlinearity increases in subsequent graphs from left to right. In the case of the minimally nonlinear model $m = 1$ the system mostly detects the input channel with the higher frequency - A_1. The weight of A_2 (orange) is only slightly higher than the weight of A_3, indicating that the CN detects the coincidence only to some limited extent. As the nonlinearity increases, the CN shows a stronger signs of the correlation since the weights are increasingly dependent on the coincidences. However, if the bolus size is increased, then again a single input is sufficient to allow the membrane potential to cross the threshold, thus the frequency biased inputs are more strongly recognised.

I will now characterise the dependence of coincidence detection on the time-delay between the correlated signals. Consider a scenario where two boli are given to the system. The first bolus A_1 comes at a fixed time and the second one follows after a time period δ. Then the accumulation of H_2 as a fraction of total weight accumulation is measured as a function of δ. Fig. 33 demonstrates that the CNs with the higher nonlinearity are more sensitive to short-term temporal coincidences, however they seem to lose the ability to recognise coincidences that are further spaced out. On the other hand, in the case of minimal nonlinearity, the differential weight update rewards the correlation less, but is not limited to detecting very small δ only. For example, for a relatively high $\delta > 0.1$, the CN with $m > 1$ does not detect any coincidences and thus the weight update is uniform. Whereas, in the case of $m = 1$ some differential weight updates can be seen for any δ.

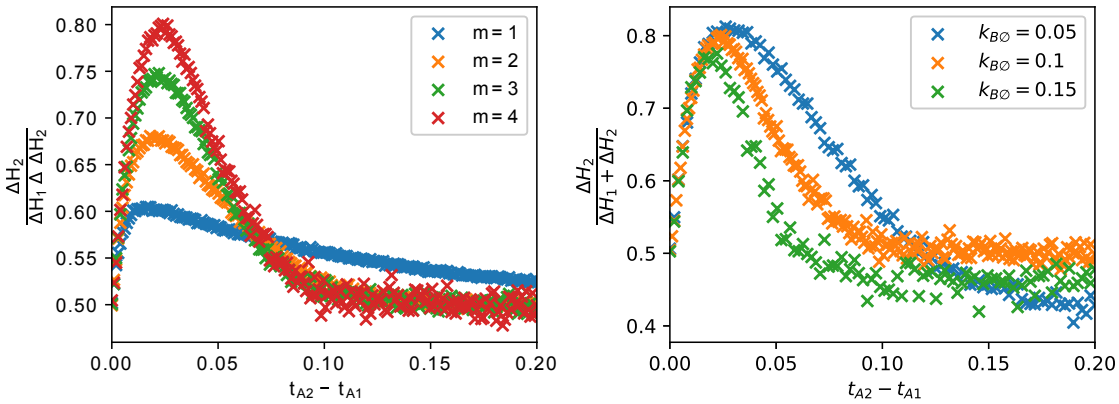
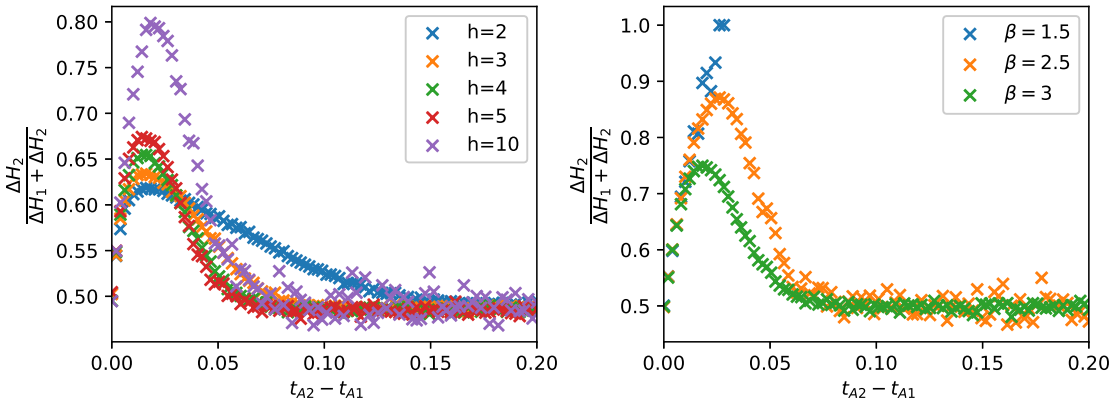


Figure 33: The left panel shows the differential weight increase for different degrees of nonlinearity. Here, 1 on the y-axis means that only the weight associated with the second channel increased, while 0.5 means that the weights of both channels were updated equally. The figure on the right shows the same value, but for different removal rates of B molecules. The faster the rate of removal, the more specific the coincidence detection. Thus, the inputs need to coincide within a narrower temporal distance. For both plots the points were computed by simulating a CN with two input channels and with initial weights of $H_1, H_2 = 0$. The neuron is provided with a bolus of A_1 at $t = 0$ and after a variable time the second bolus from A_2 is provided. The simulation then continues for another 0.2 time units. Each point on the graph is then an average over 1000 repetitions of this experiment.

In order to further explain this point, simulations equivalent to fig. 33 were conducted for a simplified model which instead of the ligand reactions employs an approximation by a Hill function or a step function, see fig. 34. Note that, these models are less biologically plausible, as the activation is calculated in a deterministic way and not implemented by a cascade of chemical interactions. A familiar pattern of differential weight updates emerges from these experiments as the Hill coefficient h is varied. For $\delta > 0.1$ the neuron with high Hill function coefficient $h = 8$ does not detect any coincidences, whereas the case of $h = 2$ shows some differential weight update throughout. In the case of a step function, the differential weight increase shows a similar qualitative trend. Notably, the neuron experiences an abrupt drop in performance when the bolus size β becomes too small to saturate the activation function.

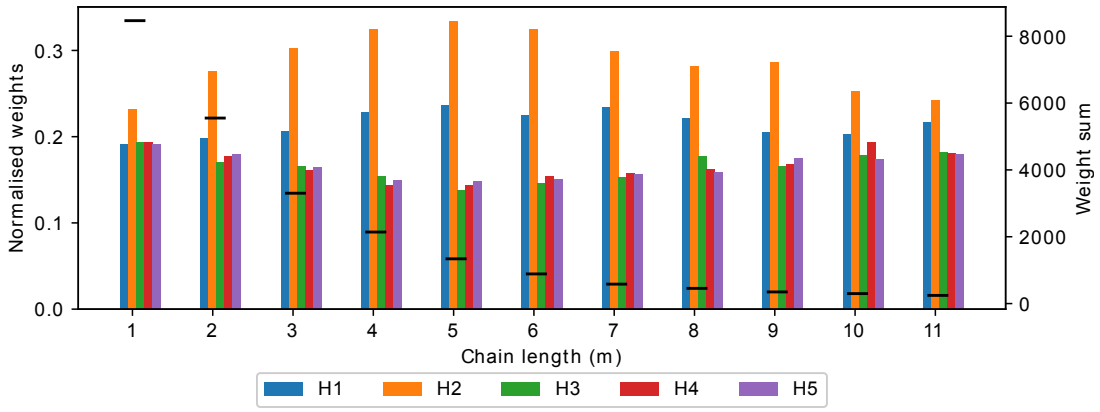


(a) (b)

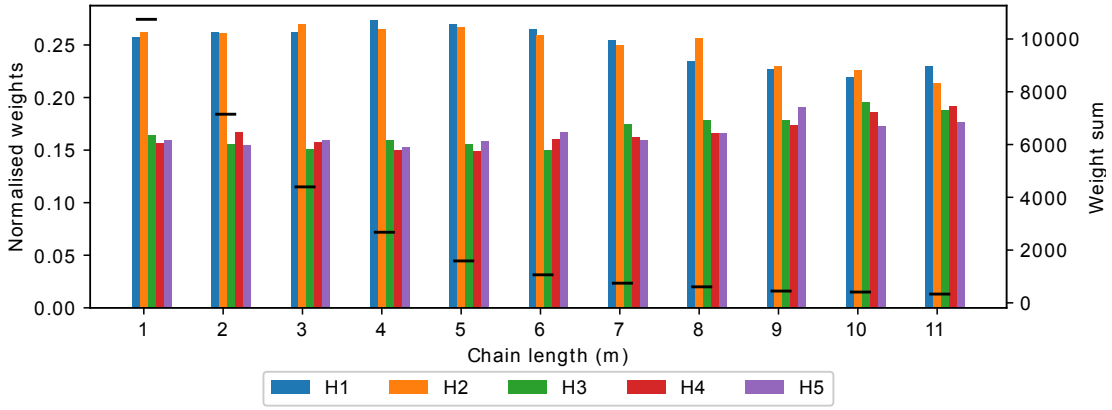
Figure 34: (a) Differential weight increase for varied steepness parameter of the Hill-function, which acts as the activation function. Same as in fig. 33, the higher value indicates a stronger learning of temporal correlations. (b) The differential weight increase for different boli sizes in a model which replaces the activation function with a step-function. Here, for sufficiently large boli ($\beta \geq 1.5$) the system learns similarly to non-deterministic systems. When the bolus is too small, it never crosses the threshold and the system doesn't learn.

4.5.2 Analysis of learning outcomes

I will now more closely examine the conjecture that the temporal correlations can be solved more effectively by the CN with higher nonlinearity. To this end, a set of CNs with $N = 5$ input channels were trained on the TC 2 and FB 2 tasks. Each of the generated CNs differs in the degree of nonlinearity of the activation function in range $m = 1, \ldots, 11$. Figure 35 reveals that changing the parameter m indeed has a great influence on the learning outcomes at the steady state. The weight sets appear to have significantly different distributions as a function of m. Whether the highest degree of nonlinearity of the activation function is optimal for learning remains unclear.



(a) Temporal correlation



(b) Frequency bias

Figure 35: Normalised weights for an (a) TC 2, and (b) FB 2 tasks as a function of chain length m. The y-axis on the right side of the figures and horizontal black bars describe the sum of weights across all input channels, which is a normalisation constant for the corresponding sets of weights. It can be noticed that the accumulation of weights becomes more difficult as the chain length increases. However, it's worth noting that both low and high m result in low variance of the weight representations.

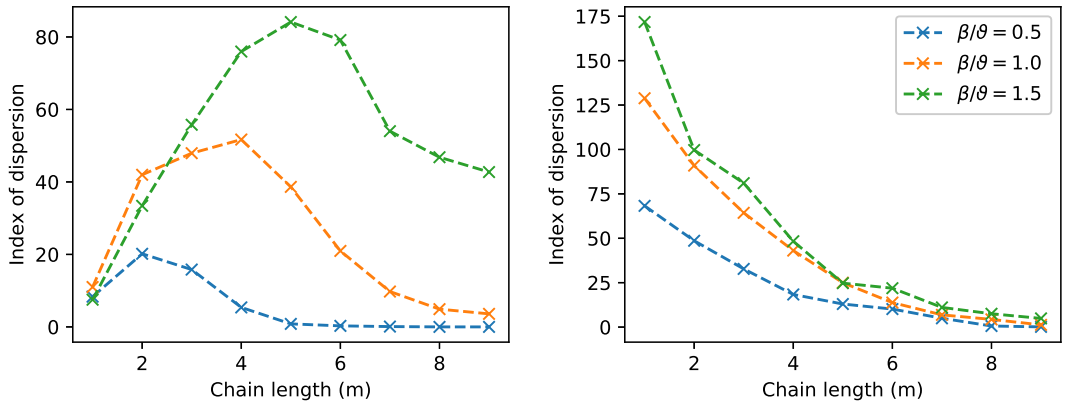
The index of dispersion, i.e. the standard deviation divided by the mean of the weights, can be used as a useful measure of the ability of the system to produce diverse weights. In simple terms, the higher the index of dispersion, the less homogeneous the weights. The assumption here is that the more diverse weight sets have an improved ability to discriminate between different biases.

The ability to discriminate between frequencies increases with the degree of nonlinearity of the activation function, which is consistently with the previously proposed hypothesis, see fig. 37. However, it does so only up to a point, which I will henceforth refer to as *the optimal nonlinearity*. Above this point the index of dispersion decreases.

The optimal nonlinearity shifts to the right, when the bolus size is increased. This is consistent with the previous results, see fig. 32. As β increases, it becomes easier to saturate the activation function and the neuron loses the ability to detect temporal coincidences. Therefore, a more steep activation function is necessary to produce optimally diverse distribution of weights at the steady state.

The above observation suggests that the decline in the performance of the CN for higher chain lengths is a result of resource starvation. This is due to the activation function which, depending on the length of the ligand chain, consumes m molecules of B from the system. As a consequence, for insufficiently large β the CN is unable to represent its internal state efficiently and the neuron fails to produce a learning signal. This effect is negligible if the maximum abundance of B is high relative to the length of the ligand chain m. This can be interpreted as a resource cost of computing nonlinearity. The higher m, the higher the bolus size required to faithfully implement the activation function.

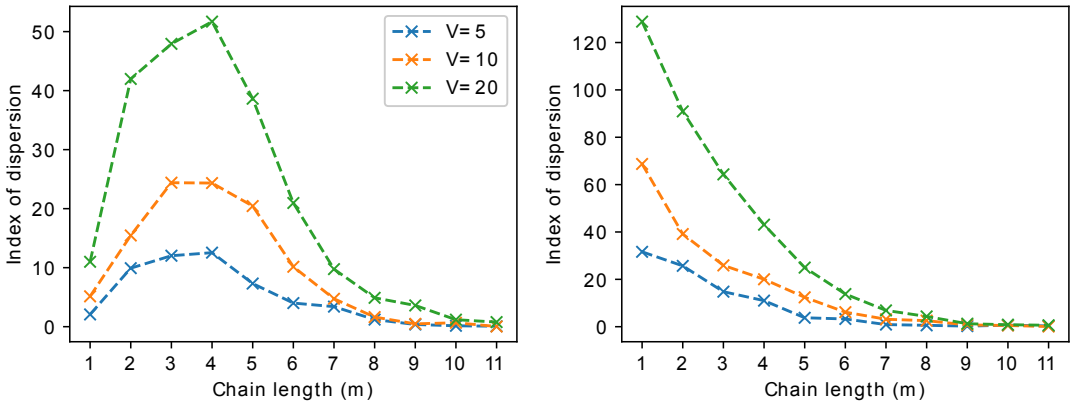
Figure 37 shows that the change in volume of the system (while keeping the bolus size β fixed) does not influence the optimal nonlinearity. This is consistent with the previous observations, and suggests that the optimal nonlinearity is primarily related to the bolus size. In any case, the change in volume still does not prevent the CN from suffering resource starvation at the higher nonlinearities ($m > 6$). In all of the previous experiments, the volume of the system was fixed at $V = 15$, which was found to allow for sufficiently stable dynamics and fast simulation. Changing the volume of the system has a twofold effect. Firstly, the amount of molecules is multiplied by the volume. This means that the inputs as well as the amount of species which remains fixed through the simulation (i.e. E_0) are increased linearly as a function of volume. At the same time,



(a) TC 2 (b) FB 2

Figure 36: Index of dispersion for the set of weights produced by the CN given bolus size $\beta = 0.5$, 1, and 1.5, for TC 2 (left) and FB 2 (right) tasks. The index of dispersion is a measure of the diversity of the weight at the steady state, thus indicating how well the CN can distinguish between the input channels. Therefore a set of weights for unbiased input would result in index of dispersion ~ 0. The graph on the left shows that there is an optimal degree of nonlinearity for detecting temporal correlations. Notably, there is a visible shift of the peak value of the index of dispersion. This is a result of the *starvation effect*. More precisely, for high values of the m parameter the system with a small bolus size is unable to compute because of the way the activation function is implemented. As the chain length increases more B molecules are consumed, and thus the neuron is unable to activate.

the volume of space in which the chemical reactions occur expands, and as a result the probability of any bi-molecular reaction to occur decreases. This is because for higher volumes the space in-between the molecules increases, and thus the collisions become less likely. Therefore their reaction rate constants needed to be divided by the current volume parameter, in order to simulate this phenomenon. Increasing the volume of the system has an effect of bringing the simulation results closer to deterministic model, which in principle can be understood as a model with infinite number of molecules. However, increasing the volume further quickly becomes expensive to simulate on a conventional computer, and thus a careful calibration of this parameter is required.



(a) TC 2

(b) FB 2

Figure 37: Index of dispersion for the set of weights of the system with $V = 5, 10, 20$ in the steady state as a function of chain length m for TC 2 task. Measuring the diversity of channel representations allows us to differentiate which m provides the learning strategy the most different from random weight allocation (IoD $= 0$). TC tasks require the activation function to be dependent on auto-correlation of inputs to a certain degree. However, for $m > 6$ the nonlinearity becomes too high and disables learning. On the other hand, FB tasks exhibit no temporal correlation of inputs, thus the index of dispersion is negatively correlated with the ligand chain length.

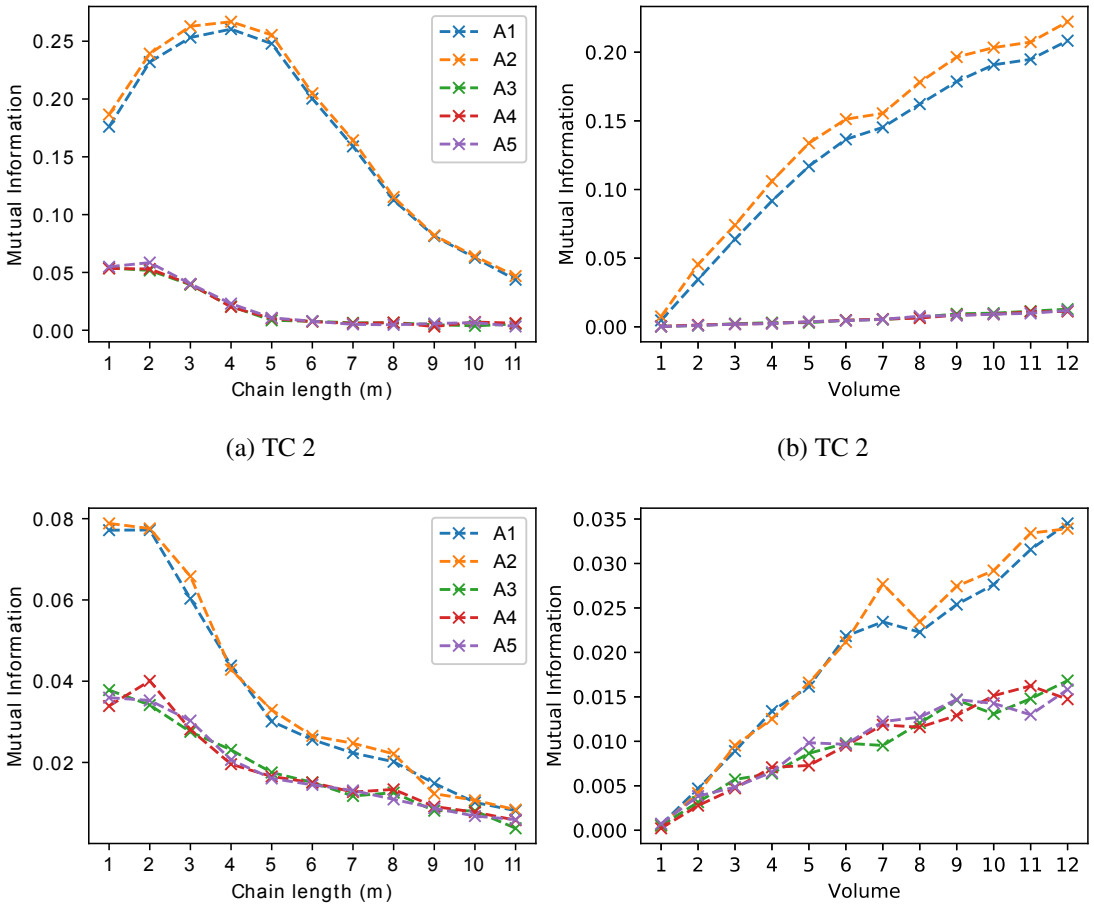
Another useful metric for measuring the performance of the trained CN is the mutual information. The mutual information is a metric which describes the amount of mutual dependence of two distributions X and Y, see eq. 39. In other words, how

much information can be deduced about one variable by observing another one.

$$\mathrm{I}(X;Y) = \int_{\mathcal{Y}} \int_{\mathcal{X}} p_{(X,Y)}(x,y) \log \left(\frac{p_{(X,Y)}(x,y)}{p_X(x) p_Y(y)} \right) dxdy \tag{39}$$

Figure 38 shows the mutual information between the distributions of molecules at the steady state for $\mathcal{E}$ and each of the input channels A_n. Molecular species $\mathcal{E}$ represents the fully occupied ligand chain, thus their presence signifies neuronal activation and catalyses the reaction responsible for the weight potentiation. Therefore, the mutual information between $\mathcal{E}$ and each of the input channels A_i reflects their influence on threshold crossing events, and thus synaptic learning.

Plotting mutual information confirms the findings from the previous experiments. Figure 38 shows a similar qualitative pattern in terms of optimal nonlinearity to experiments in figs. 36 and 37. Noticeably, the mutual information also tends to drop as the system approaches the resource starvation. Moreover, an interesting picture is revealed when looking at biased and unbiased inputs separately. The biggest difference in mutual information of these sets of weights also seems to occur at the point of optimal nonlinearity. This shows that in the case of temporal correlation tasks not only the biased synapses have the highest mutual information at $m = 4$, but also the unbiased ones show signs of decline. Therefore, the separation between those two sets of weights is the greatest here, for the case of learning temporal coincidences. In the case of frequency bias, similarly the biggest difference in the mutual information is at $m = 1$, which confirms the observations from fig. 37 and 36.



(a) TC 2

(b) TC 2

(c) FB 2

(d) FB 2

Figure 38: Mutual information between the activation function ($\mathcal{E}$) and an input channels (A_n) after training on TC as a function of (a) chain length for fixed $v = 20$, and (b) volume for fixed $m = 4$. The amount of information about the neuronal activation carried by an input channel differs immensely between biased (A_1 and A_2) and unbiased synapses. Again, it's noticeable that a degree of dependence on temporal auto-correlation in neural activation is needed for more efficient learning of TC tasks. The amount of useful information captured by the weights peaks at $m = 4$. Moreover, it can be observed that the volume of the system scales the mutual information logarithmically.

4.5.3 Measuring the thermodynamical cost of computation

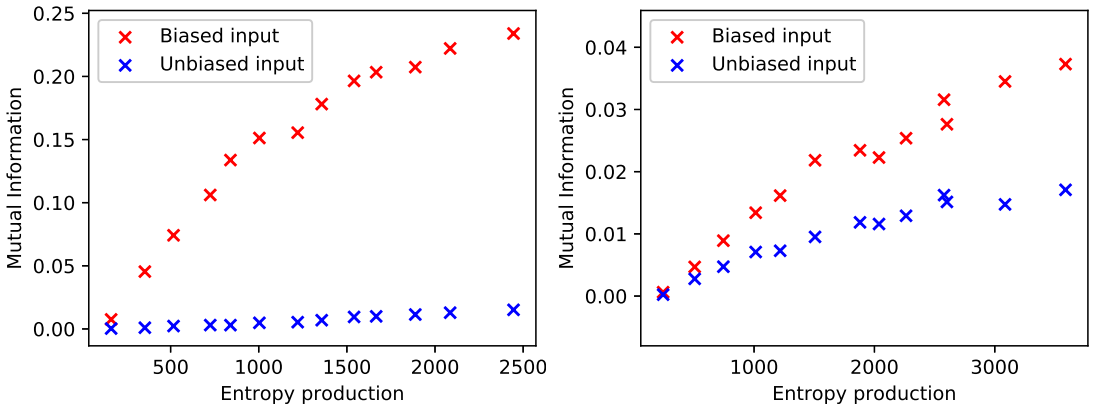
The energy efficiency of neural systems has been a widely researched topic in recent years, in particular in the field of stochastic thermodynamics (Valadez-Godínez, Sossa

and Santiago-Montero 2020). Here, I will identify the energy cost of computation with the entropy produced during the computation. Entropy production is a measure of cost associated with irreversible processes (Bennett 1982; Seifert 2012), and has been shown to be related to the thermodynamical cost of such reactions (Tomé and de Oliveira 2018; Goldt and Seifert 2017).

The entropy changes each time the system transitions from one state to the next (i.e. any reaction occurs), since each transition results in heat dissipated into the environment. Notably, the entropy may also decrease, however it does so with a much lower probability. Seifert (2005) shows that this can be approximated by the ratio of the forward rate of the state transition w_{ij} and the respective backwards rate w_{ji}:

$$\Delta S = \ln \frac{w_{ji}}{w_{ij}} \tag{40}$$

As the CN learns and its weights approach a steady state, the entropy production rate of the system also reaches a steady value. Figures 39 shows the tradeoff between the cost of computation in terms of entropy production at the steady state and the mutual information between the learning signal and certain input channels (as defined in section 4.5.2). Here, the volume of the system is varied from 1 to 12, in order to examine a trade-off between the cost of computation and quality of learnt representations. For small systems with volume close to 1, there are not enough molecules in the environment, thus the noise around the solution is high. This results in a negligible difference in mutual information between the biased and unbiased input channels. However, as the volume increases, the mismatch between them widens. This trend appears to saturate as the volume approaches infinity, and thus more closely approximates a deterministic model.



(a) TC 2

(b) FB 2

Figure 39: Tradeoff between the cost of computation in terms of entropy production and mutual information for tasks (a) TC 2 and (b) FB 2. Here, the volume of the system was varied, and the degree of nonlinearity was set to $m = 4$ for TC task, and $m = 1$ for FB. The post-spike reset parameter was fixed for all experiments at $\beta = 1$.

4.6 Interpreting CN as a single-celled organism

The CN model discussed so far is consistent with regards to mass-action kinetics, however it does not meet the criteria for biochemical plausibility. For example, species A, B, and H would have to be interpreted as conformations of the same molecule with different energy levels. On top of that, those conformations would need to have specific enzymatic properties. In this section, I will explore viable options to improve the biological realism of the CN model. I will propose a plausible molecular interpretation of necessary neuronal mechanisms, explaining how computation can be done in an environment of single-celled organisms.

I will now explain the biochemical version of the CN in more detail (see tables 4 and 5). This version of the CN model is split into a number of compartments representing different input channels, see fig. 40. In the basic model, the indexed species (such as A_i and H_i) referred to different species that exist in the same volume, while in this case

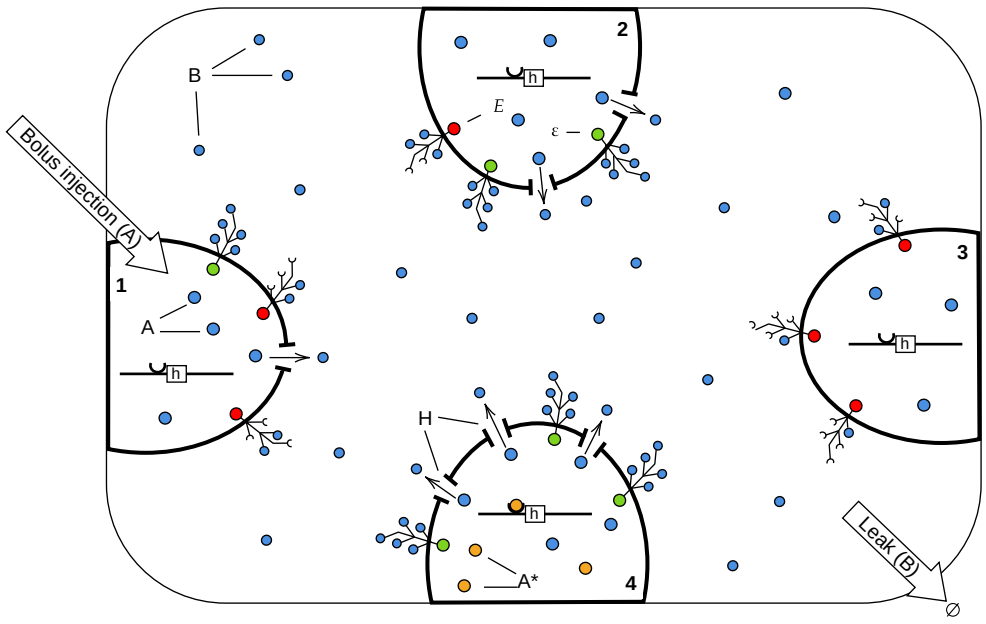


Figure 40: Graphical representation of a compartmentalised version of the CN. Here, the molecules representing inputs to different channels A_i and A_j are the same molecular species but contained in different compartments i and j respectively. The activated form of the input species - A^* binds to the promotor site of gene h and activates its expression. The weight species H act as an active transporter molecules for A. The membrane potential molecules B are the same as any of the input species A_i when transported to the outermost compartment. Each of the compartment has a trans-membrane protein E with m extra-cellular binding sites. When all of the m binding sites are occupied by B molecules, then the internal site becomes active (indicated by green). This in turn catalyses the activation of A which becomes A^*.

they are interpreted as the same type of molecules in different compartments. Therefore, different A_i would then be the boli of the same molecular species injected into a compartment i. In turn, these compartments are contained within another enclosed environment or "extra-cellular space". The individual input channel compartments can be thought of as individual bacterial cells or artificial membranes with a minimal genome.

In this interpretation, the molecules of type H_i are interpreted as transporters for A_i molecules. The conversion of A_i to B is now interpreted as export of A_i from compartment i to the extra-cellular space. The molecular species B is then the same as A_i but contained directly in the outer compartment, and the process of moving these

between the compartments is facilitated by the H_i species. Without the H_i species, there would be no B molecules in the system, since the A_i species would have no way to leave their compartment i.

The rate of export of A_i is specific to each compartment in that it depends on the local abundance of H_i. To regulate this, molecules H_i are assumed to be expressed by a gene h. This means that H_i molecules are produced when the gene is activated. Here, the genes can be understood as binary switches which act slowly compared to the other reactions. They become activated as a result of certain molecules binding to their promoter sites.

The E molecules are interpreted as transmembrane proteins that are embedded in the membrane of each compartment. This means that each of these proteins is partially contained within the compartmentalised channels (intra-cellular part), while the remaining fragment is on the outside of the membrane separating the compartments from the outside (extra-cellular part). Their extra-cellular site has m binding sites for B molecules which bind cooperatively, i.e. the binding properties change as more sites are occupied. When all sites are occupied then the intra-cellular part is activated, i.e the transmembrane protein enters its active state $\mathcal{E}$. In this model, there also exists an activated form of A_i, denoted by A_i^*. The basic model does not have an analogue of this activated form of A_i. The conversion of $A \rightleftharpoons A_i^*$ is mediated by the activated form of the transmembrane protein $\mathcal{E}$.

Each compartment contains a gene h that codes for the molecule H_i. Expression of the gene is triggered by the activated form of A_i. Here, A_i^* binds to a promoter site of gene h_0 and in turn activates it. This means that the gene now becomes its active form h, and starts to produce the molecular species H_i. There also is a low leak expression by the unactivated gene, which is denoted as k_{leak} in tables 4 and 5. This results in a constant low supply of H_i, which is needed as a catalyst in the input integration reaction. Gene activation of this type is frequently modelled using Michaelis-Menten kinetics, and thus it is a good approximation of the corresponding enzyme kinetics in

Function	Reaction
Input	$\mathrm{I}_n \underset{\mathrm{k_{AI}}}{\overset{\mathrm{k_{IA}}}{\rightleftharpoons}} \mathrm{A}$
	$\mathrm{A} + \mathrm{H} \underset{\mathrm{k_{HA}}}{\overset{\mathrm{k_{AH}}}{\rightleftharpoons}} \mathrm{AH} \underset{\mathrm{k_{BH}}}{\overset{\mathrm{k_{HB}}}{\rightleftharpoons}} \mathrm{B} + \mathrm{H}$
Activation function	$\mathrm{B} + \mathrm{E}_i \underset{\mathrm{k^-}}{\overset{\mathrm{k^+}}{\rightleftharpoons}} \mathrm{E}_{i+1}, \qquad i < m-1$
	$\mathrm{B} + \mathrm{E}_{m-1} \underset{\mathrm{k^-_{last}}}{\overset{\mathrm{k^+}}{\rightleftharpoons}} \mathcal{E}$
Weight accumulation	$\mathcal{E} + \mathrm{A} \underset{\mathrm{k_{EA}}}{\overset{\mathrm{k_{AE}}}{\rightleftharpoons}} \mathcal{E}\mathrm{A} \underset{\mathrm{k_{EA^*}}}{\overset{\mathrm{k_{A^*E}}}{\rightleftharpoons}} \mathcal{E} + \mathrm{A}^*$
	$\mathrm{A}^* \underset{\mathrm{k_{AA^*}}}{\overset{\mathrm{k_{A^*A}}}{\rightleftharpoons}} \mathrm{A}$
	$h_0 + A^* \underset{\mathrm{hA^*}}{\overset{\mathrm{A^*h}}{\rightleftharpoons}} \mathrm{h}$
	$h_0 \xrightarrow{k_{\mathrm{leak}}} \mathrm{H}_n + \mathrm{h}_0$
	$h \xrightarrow{k_h} \mathrm{H}_n + \mathrm{h}$
Leak	$\mathrm{H} \xrightarrow{\mathrm{k_{H\varnothing}}} \varnothing$
	$\mathrm{B} \xrightarrow{\mathrm{k_{B\varnothing}}} \varnothing$

Table 4: List of chemical reactions in a single CN unit interpreted as a cell. Molecular species $A, E, \mathcal{E}, h_0, h$ and H are compartmentalised. Each compartment has a gene h_0 which when activated by A^* can express a transporter H.

Function	Reaction rates
Input	$k_{IA} = 10$, $k_{AI} = 0.000001$
	$k_{AH} = 0.03$, $k_{HA} = 0.000001$, $k_{HB} = 100$, $k_{BH} = 0.000001$
Activation function	$k^+ = 1$, $k^- = 5$
	$k^-_{last} = 0.5$
Weight accumulation	$k_{AE} = 0.2$, $k_{EA} = 0.000001$, $k_{A*E} = 0.2$, $k_{EA*} = 0.000001$,
	$k_{A*A} = 0.05$, $k_{AA*} = 0.000001$
	$k_{A*h} = 1$, $k_{hA*} = 0.1$
	$k_{\mathrm{leak}} = 0.0001$
	$k_h = 1$
Leak	$k_{H\varnothing} = 0.0003$
	$k_{B\varnothing} = 0.1$

Table 5: List of reaction rate constants in a biochemical interpretation of the CN model.

the basic model of the CN.

A major difference between this version of the CN and the one introduced earlier is that the molecules E are now specific to each compartment. Thus, the minimum number of instances of E is N, while in the basic model a single copy of E at time $t = 0$ could be sufficient. One consequence of this design choice is that at any particular time the number of occupied binding sites will typically be different across the different N compartments. This is a source of additional variability. Moreover, since the number of copies of E is higher than in the basic CN (see table 3), it becomes more susceptible to the resource starvation of B as a result of the extra-cellular binding sites withdrawing molecules from the outer compartment. However, both of these issues can be solved by tuning the model in a way that the abundance of B molecules is high in comparison to that of E. This shows that the resource starvation problem becomes even more severe in this case, and therefore the bolus size needs to be further increased, in order to accommodate for the same degree of nonlinearity.

4.6.1 Performance analysis

The performance of this biological interpretation of the CN was tested on the task of associative learning. As discussed before, Fernando et al. (2009) has proposed a biochemical model which implemented a form of Pavlovian learning. However, it had multiple shortcomings and was not scalable to more input channels (see section 2.6.2). Figure 41 shows that the proposed biochemical system has the same capacity for associative learning as the original CN model.

This biochemical version of the CN is also capable of learning in a variety of tasks involving frequency bias and temporal correlations. The results presented in fig. 42 show a similar qualitative pattern to the original CN model (see fig. 31). Although the index of dispersion for the obtained sets of weights is higher in the case of the original design, different types of bias embedded in the input data are still clearly recognised.

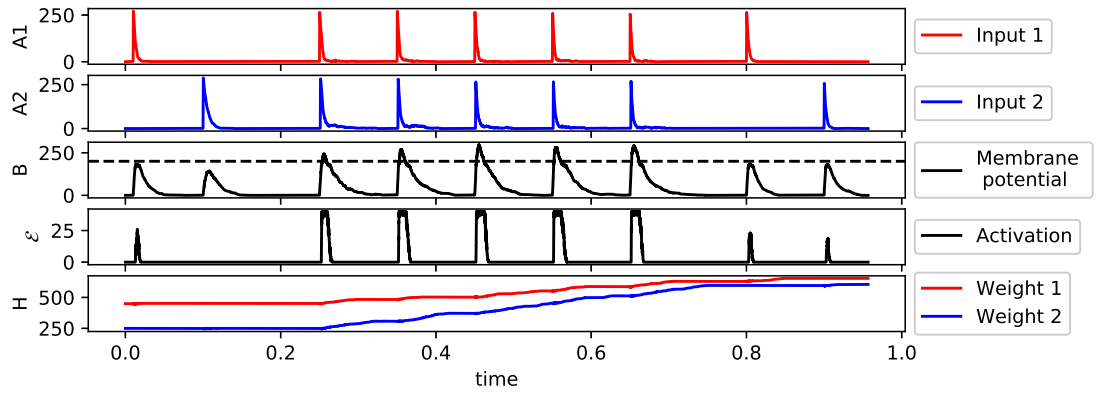


Figure 41: Example of associative learning in cellular interpretation of the CN. The task is the same as in fig. 29, where the neuron is presented two inputs. At first, $\mathcal{E}$ molecules are only produced in response to the first input, and none are produced when a signal from the second one is presented. Next, the CN is exposed to both inputs in close temporal proximity multiple times. The fourth graph shows that the Weight 2 grows each time the inputs coincide, however Weight 1 has saturated at this point. Lastly, at time 0.9 the CN is presented with the Input 2 only. After just a few examples CN is able to respond in reaction to Input 2 without any additional stimulation.

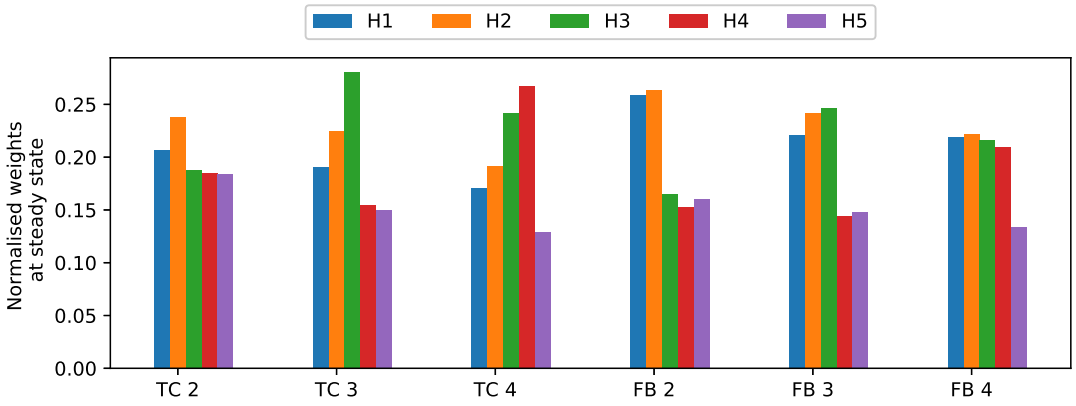


Figure 42: Normalised weights for a variety of TC and FB tasks. The nonlinearity or the chain length is set to $m = 4$ for the TC, and $m = 1$ for FB, the same as in fig. 31. The CN is trained for 3 tasks with temporal correlation of inputs, where 2, 3 or 4 synapses are biased respectively. Similarly for FB, where 2, 3 and 4 synapses have the higher spiking frequency respectively. The weight distributions at the steady state proves to be highly correlated with the type of bias embedded in the data.

4.7 Chapter summary

The main contribution of this chapter is the chemical neuron model (CN), which is the first fully autonomous design for a biochemical system which can learn spatio-temporal patterns. The proposed design mimics a spiking neuron, and its learning algorithm produces results similar to those of a spiking neuron with STDP. Unlike some of its predecessors, such as (McGregor et al. 2012; Fernando et al. 2009; Blount et al. 2017), this model is fully autonomous in learning and can be arbitrarily scaled to accommodate for more input channels. Two versions of the CN were discussed. One which is readily interpretable as a chemical reaction network, and the other which is more biochemically plausible. The former one, provides a framework for understanding minimal energy costs associated with neuronal computation. Thanks to this, it is possible to estimate the computational cost as measured by the entropy production. The second cellular model, on the other hand, is built using well-known biochemical motifs. Thus it describes a system which in theory could be implemented in nature. Although the model would still be difficult to synthesise with current technology, it uses well defined components found in living organisms such as: gene expression, cooperative binding, and transmembrane proteins.

Another contribution of this chapter was the establishment of criteria for *autonomy*, see section 4.1. These criteria are not only necessary for building theoretical systems which can perform certain intelligent functions, but also ensures that it can learn without supervision and that all of the functions are embedded into the system. This means that both the learning algorithm, as well as all other functions need to be internal to the model itself. These considerations are crucial when considering the experimental implementation in the context of synthetic biology. Biological systems cannot be stopped in order to readjust the weights, and therefore they require the learning algorithm which works without an outside interference.

The investigation conducted in this chapter has shown that the degree of nonlinearity, controlled by parameter m allows for better learning of temporal correlations.

However, even models with $m = 1$ (the least nonlinear model) still can perform reasonably well, and can distinguish inputs which spike in a synchronised way. This is consistent with the findings described in Chapter 3, showing that the temporal autocorrelation of the membrane potential is the key parameter for learning, while the spikiness of the model was found to be less important.

The CN consists of chemical reactions following mass-action kinetics, and therefore allows for the estimation of entropy production. I evaluated the cost of computation with regards to the scale of the system, as well as the degree of nonlinearity in the activation function. More in-depth analysis has shown that the CN experiences a resource starvation of B species, as the parameter m increases. This means that the models which more closely approximate spiking activation require increasingly more resources. Although higher m allows for better performance in the tasks which involve learning of temporal correlations, when the nonlinearity is sufficiently high, the neuron loses the ability to learn and react to stimuli correctly. The starvation effect can be partially solved by means of increasing the bolus size relative to the threshold, see fig. 36. This, however, comes with an extra cost in terms of entropy production. These additional costs come from two sources. First of all, more particles need to be added to the system at each input. Secondly, perhaps more importantly, the additional expenditure comes from the programmed decay of H and B molecules outside of the system. This discovery brings forth the question whether such nonlinear functions pose an additional cost in neuronal systems. Although outside of the scope of this chapter, this investigation could be an interesting point for further research on the tradeoff between metabolic cost and the neuron's ability to compute temporal patterns.

Chapter 5

Neural modelling in DNA-strand displacement

5.1 Introduction

In the previous chapter, I have demonstrated that, in principle, autonomous neuromorphic learning systems could be implemented as networks of chemical reactions. Although the chemical neuron is autonomous and biochemically sound, it would be difficult to synthesise in a laboratory. This is mainly because the synthesis of proteins with these very specific properties still remains impossible. In fact, it is unclear whether such proteins exist at all.

One promising approach is to employ gene regulatory networks (Racovita and Jaramillo 2020). Fernando et al. (2009) demonstrated that a relatively simple gene regulatory network (GRN) can implement a basic version of associative learning. This system, however, learns an association after just a single coincidence of inputs and is unable to forget the association between already learnt inputs. Moreover, it is limited to recognising the correlations between two input channels only. This makes it unsuitable for solving more complex tasks, such as detecting temporal correlations in a multichannel stream of inputs. Other researchers employed evolutionary algorithms, in order

to produce GRNs capable of associative learning (McGregor et al. 2012). More biologically plausible systems comprised of multiple compartments were also proposed (Macia, Vidiella and Solé 2017a; Macia and Sole 2014; Macia, Vidiella and Solé 2017b). However, they too lack the autonomy and are not suitable for learning spatio-temporal data.

Reliable programming of GRNs still remains a challenge due to a number of reasons:

1. Cross-talk is a common problem (Grah and Friedlander 2020; Grant et al. 2016). Due to non-specific binding between biochemical molecules the behaviour of the system is unpredictable.

2. Gene regulation occurs on multiple levels, such as gene expression, post-translational control, or protein degradation. This regulation also acts on multiple scales, it can only affect a specific gene or act globally (Yubero and Poyatos 2020).

3. Lastly, these regulatory mechanisms are likely to pose a metabolic burden to the host organism. This would result in evolutionary pressure to remove such a circuit, if it doesn't bring enough benefits to the host (Wang, Wei and Smolke 2013).

One alternative to gene-regulatory networks could be DNA-based computing. In particular, DNA strand displacement (DSD) (Lakin et al. 2011) has a potential to be a reliable substrate for computation. Most importantly, this framework allows for the behaviour of real DNA machines to be accurately predicted from simulation (Yurke et al. 2000; Fontana 2006). Notably, a framework of two-domain DSD (Cardelli 2010) allows for building DNA circuits which are efficient and free of cross-talk, which is often associated with biochemical systems (Grah and Friedlander 2020). Moreover, Chen et al. (2013) have demonstrated that arbitrary chemical reaction networks can be implemented in DSD.

DSD has been used to implement a vast array of intelligent behaviours, such as linear-threshold circuits and logic gates (Seelig et al. 2006; Qian and Winfree 2011), switches and oscillators (Dalchau et al. 2018; Lakin et al. 2012), consensus algorithms (Chen et al. 2013). DSD cascades have also been used to analyse cells by classifying their surface markers (Rudchenko et al. 2013). Neuronal systems implemented in DNA have long been of interest to researchers. Early attempts include a Hopfield network implemented through DSD cascades (Qian, Winfree and Bruck 2011). Although the network was shown to be capable of recreating partially erased patterns, the weights were obtained from separate simulations and were hard-coded into the system. Other researchers recently attempted to design competitive winner-take-all networks of perceptron neurons (Genot, Fujii and Rondelez 2013; Cherry and Qian 2018). This approach allows the DSD network to perform multi-class learning, which was shown to be useful for example in the recognition of handwritten digits from the MNIST dataset (LeCun and Cortes 2010). However, similarly to the previous attempts, the learning phase was performed on a digital computer, and then the obtained weights were transferred onto the DNA circuit.

Some researchers attempted to build DSD systems which can learn without the need for digital computers. For example, Lakin and Stefanovic (2016) demonstrated that a form of gradient descent learning could be constructed through the DSD interactions. They used a two-concentration multiplier circuit motif, and demonstrated that a neuron can adjust its weights internally given an external feedback. However, this means that this network also requires constant feedback from an external observer, and thus does not meet the criteria for being autonomous.

In this chapter, I will present a DNA-based design for a spiking neuron capable of autonomous Hebbian learning - the DNA neuron. The DNA circuit proposed here is based on an approach presented in the previous chapter: the chemical neuron. Similarly to the chemical neuron, this system also meets all necessary criteria for being autonomous (see section 4.1). This means that the neuronal model, synaptic weights, as

well as the learning algorithm are implemented within the reaction network. Therefore, the system does not require any external agent to read its state, adjust the weights, or implement the signal modulation. The system's learning algorithm functions in a similar way to that of a spiking neuron with STDP. This DNA neuron model is based on an experimentally proven design motif of two-domain DNA strand displacement (Cardelli 2010), which allows for cross-talk free implementation of DNA-based systems. This design is scalable to an arbitrary number of input channels, and can readjust its weights based on new learning episodes. Through extensive simulations, I will demonstrate that the DNA neuron can recognise temporal and statistical biases embedded within noisy streams of sensory data, in the sense that the steady state abundance of specific weight molecules reflect the biases of its inputs.

5.2 DNA neuron implementation in CRN

All of the interactions between the molecules constituting the neuron are facilitated by four catalytic reactions (see fig. 43 and table 6). The DNA implementation of a spiking neuron, that I will describe below, is similar to the one proposed in the previous chapter, however it will include a few crucial differences. The chemical reaction network (CRN) discussed in this section is a special case of a *chemical neuron*, with a simplified activation function (i.e. $m = 1$). Moreover, the four key interactions implementing the CN now become catalytic reactions. Firstly, I will briefly describe this system as a CRN, before showing, in the next section, how to translate this to a system of DNA-strand displacement interactions.

Signal integration

The inputs are instantaneously injected into the system via N concurrent input channels. It is assumed that input channel n receives input in the form of molecules of type A_n, at a particular time t_n^s. A_n molecules from each channel facilitate the conversion of

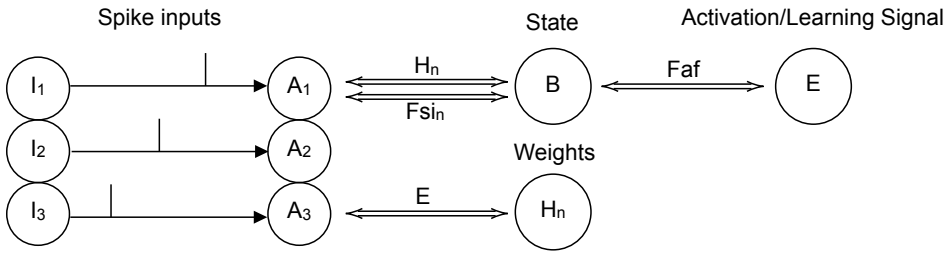


Figure 43: Graphical representation of an autonomous neuron implemented as a chemical reaction network. The inputs to the system are provided in the form of discrete spikes of molecular species A_n. The double arrows signify catalytic reactions, which can be implemented by a combination of two-domain Join and Fork gates in DSD.

Function	Reaction
Signal integration	$\mathrm{Fsi}_n + \mathrm{A}_n \rightleftharpoons \mathrm{A}_n + \mathrm{B}$
Activation function	$\mathrm{B} + \mathrm{Faf} \rightleftharpoons \mathrm{Faf} + \mathrm{E}$
Weight accumulation	$\mathrm{A}_n + \mathrm{E} \rightleftharpoons \mathrm{E} + \mathrm{H}_n$
Signal modulation	$\mathrm{A}_n + \mathrm{H}_n \rightleftharpoons \mathrm{H}_n + \mathrm{B}$

Table 6: List of reactions in a single DNA neuron implemented as a chemical reaction network.

the respective fuel molecules Fsi_n into B. Molecular species B represents the internal state (or the short-term memory) of the neuron.

Activation function

Molecular species E is used to implement the activation function and the learning signal, i.e. learning only takes place when abundance of E is high. B molecules are converted into E by the catalyst Faf.

As can be seen from table 6, E is only produced when the state of the neuron reaches a sufficient level. Depending on the amount of inputs accumulated within a time window, more or less of the activation is exhibited. The coincidence of E and A_n molecules triggers learning.

Weight accumulation

The process of learning is understood as the accumulation of channel specific H_n molecules. These species can be interpreted as synaptic weights of the neuron and they play the role of a long-term memory. Functionally they impact the dynamics of the system in that they act as catalysts in the signal modulation reaction.

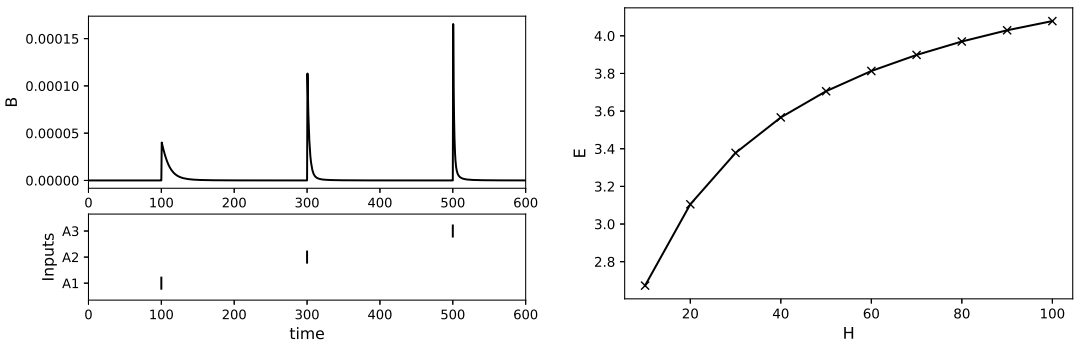
Signal modulation

Depending on the current abundance of weight molecules H_n, the speed of conversion of A_n to B changes. Figure 44 shows the result of different weight abundances on the state of the neuron (B), and the activation (E). The signal modulation reaction: A_n + H_n $\rightleftharpoons$ H_n + B, has a twofold effect. Firstly, it promotes highly weighted channels by increasing their immediate effect on the state of the neuron. This results in a higher peak value of B when an input from these channels arrives. Secondly, by increasing the rate of conversion of A_n, the subsequent weight updates become smaller. This achieves a self-regulatory mechanism similar to the well known Oja's rule (Oja 1982). As a result, the neuron's weights have a limited range of values they can take, and they don't grow infinitely.

Additionally, there are reactions which allow for a constant decay of E and H_n molecules outside of the system. This in turn, enables the system to reach a homeostatic state after a learning period, and given continuous inputs.

5.3 Neuron implementation in DSD

Soloveichik, Seelig and Winfree (2010) have shown that the molecular circuits described by CRNs can be closely approximated by DSD circuits. In particular, two-domain strand displacement has promised an intuitive framework for building efficient and cross-talk free implementations (Cardelli 2010; Chen et al. 2013). Restricting computation to two-domain strands helps to protect against unexpected interactions



(a) Internal state (B) as a function of H_n.

(b) Activation (E) as a function of H_n.

Figure 44: The internal state and activation of the DNA neuron as a function of differently weighted inputs. (a) 5 μM of A_n molecules are injected into the system at $t = 100$, 300, and 500. Each of the three channels has a different amount of weight molecules associated with it: $H_1 = 10$, $H_2 = 50$, and $H_3 = 100$. Different amount of weights results in different conversion rates from A_n to B. This means that the effective reaction rate of A_1 is slower than that of A_2, which is in turn is slower than the conversion of A_3. (b) The influence of differently weighted inputs on the concentration of E species. The DNA neuron shows a different amount of activation as a function of the weights. Here, a constant decay of E and H_n species is assumed with the rate constants of $k_E = 0.1$ and $k_H = 0.00002$ respectively.

between single stranded species, which can occur with more complex molecules. Also, as all double-stranded structures are stable, and can only change once a single-stranded component has bound, there is no possibility for gate complexes to polymerise and interact with each other. In the two-domain DSD scheme, each signal species comprises of a short domain, or *toehold*, of between 4 to 10 nucleotides, which can bind and unbind from complementary strands, and a long (recognition) domain, which are at least 20 nucleotides in length, and are assumed to bind irreversibly. In the following sections, the standard syntax of the Visual DSD programming language (Lakin et al. 2011) (see section 2.6.3) will be used to describe the species present in the system.

All of the species that facilitate computation in the neuron are two-domain DNA strands. The four main two-domain strands that enable communication between different modules of the DNA neuron are shown in table 7. Henceforth, I will refer to a short domain of a two-domain DSD strand A_n as `ta` and its corresponding long domain as

Name	Signal	DSD Species
Input	A_n	<ta^ an>
Weights	H_n	<th^ hn>
Internal state	B	<tb^ b>
Activation	E	<te^ e>
Activation function fuel	Faf	<tfaf^ faf>
Signal integration fuel	Fsi_n	<tfsi^ fsin>

Table 7: List of key DNA strands which facilitate learning.

Signal Species	Toehold	Bind	Unbind
A_n	ta	0.01	10
H_n	th	0.01	10
B	tb	1	10
E	te	0.05	10
Fsi	tfsi	1	10
Faf	tfaf	1	10
Ism	tism	1	10
Iaf	tiaf	1	10
Isi	tisi	1	10
Iwa_n	itwan	1	10

Table 8: List of toehold domains and their respective binding and unbinding rates in μMs^{-1}.

an, where n is a channel index. Therefore, the recognition of each input and weight strand is dependent on their long domains, rather than their toeholds. The same convention will be used for all other channel specific two-domain DNA species. For the detailed description of the nucleotide structure and binding rates, see tables 8 and 9.

To implement CRNs using two-domain DNA strand displacement, a collection of partially double-stranded gates can be used (Cardelli 2010; Chen et al. 2013). Accordingly, the reactivity of the DNA neuron signal strands is described by appropriately designed gate complexes, which allow for binding of the reactants (fig. 45).

For each reaction, a *Join* gate is responsible for binding of two *reactants*. In turn the Join gate produces a translator strand which signifies that the reaction has successfully

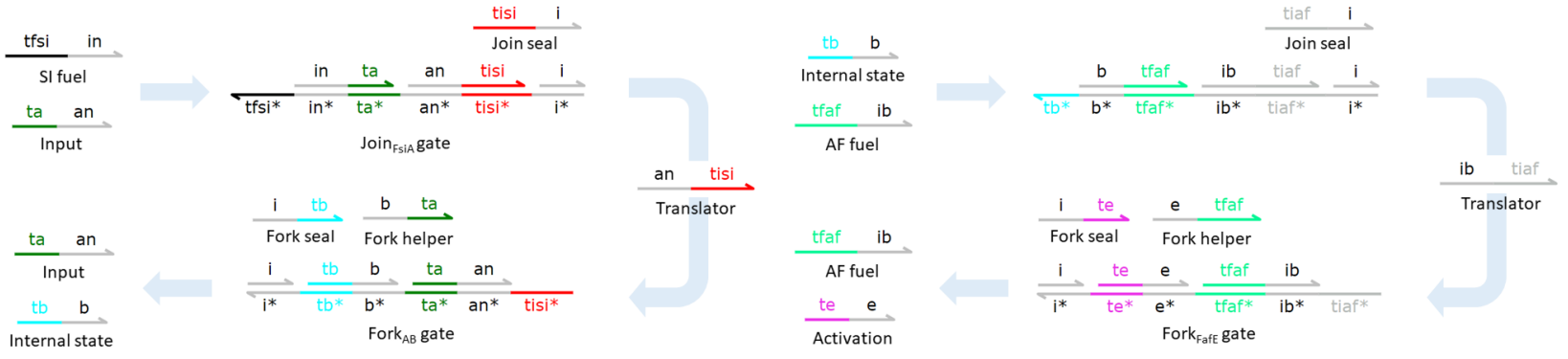
Strand	Sequence
<ta^ a1>	$TACCAA + CCCTTTTCTAAACTAAACAA$
<ta^ a2>	$TACCAA + CCCTTATCATATCAATACAA$
<ta^ a3>	$TACCAA + CCCTATTCAATTCAAATCAA$
<th^ h1>	$TATTCC + CCCTTTACATTACATAACAA$
<th^ h2>	$TATTCC + CCCTTTACATTACATAACAA$
<th^ h3>	$TATTCC + CCCTTTACATTACATAACAA$
<tb^ b>	$ACTACAC + CCCTATTCAATTCAAATCAA$
<te^ e>	$GCTA + CCCTTATCATATCAATACAA$
<tfsi^ fsi1>	$CCTACG + CCCTAAACTTATCTAAACAT$
<tfsi^ fsi2>	$CCTACG + CCCTATACTATACAATACTA$
<tfsi^ fsi3>	$CCTACG + CCCATTACTAATCAATTCAA$
<tfaf^ faf>	$CCCT + CCCATAACTATTCTAAACTA$
<tisi^>	$TCTCCA$
<tism^>	$CCCTA$
<tiaf^>	$GACA$
<tiwa1^>	$GTCA$
<tiwa2^>	$TACCAA$
<tiwa3^>	$CATCG$
<i>	$CCCTTAACTTAACAAATCTA$

Table 9: List of the two-domain DNA strands and their respective nucleotide sequences in the DNA neuron. The suitable nucleotide sequences were produced using the Microsoft Visual DSD software.

completed. Then, the translator activates a *Fork* gate, which is responsible for releasing the reaction *products*. Additional energy must be supplied to completely release all products from Fork gates, as the translator strand will only displace the first product. Appropriately designed *helper* strands are therefore placed in solution to release subsequent products, after the first product has unbound leaving an exposed toehold. As has been employed previously (Chen et al. 2013), the original design as proposed by Cardelli (2010) is extended by incorporating an additional long domain on the left-hand side of the Fork gate, which upon binding of an appropriate auxiliary molecule, *seals* the gate to prevent rebinding of its outputs. Here, all of the Join gates are also extended in an equivalent way, thus preventing rebinding of the translator strand.

In order to understand the equivalence of the CRN and the DSD implementations, let's consider the reaction $\text{Fsi}_n + \text{A}_n \rightleftharpoons \text{A}_n + \text{B}$ (fig. 45a) as an example of all 4 other catalytic reactions in the DNA neuron. A $\text{Join}_{\text{AFsi}}$ gate is defined by a double stranded complex that enables the binding of two reactants: Fsi_n and A_n. It only releases the translator strand when both of these have bound to the structure. Firstly, the fuel species Fsi_n binds and displaces the incumbent bound `<in ta^>` molecule, exposing the `ta^` toehold. This allows for the binding of A_n (`<ta^ an>`), which in turn displaces the translator strand: `<an tisi^>`. The release of this strand is a sign that both of the reactants have bound to the complex. The $\text{Join}_{\text{FsiA}}$ gate is then sealed by the binding of `<tisi^ i>`, which prevents rebinding of the translator. This reaction results in the release of the seal waste molecule: `<i>`.

The Fork_{AB} gate is then triggered by binding of the translator strand released by the corresponding Join gate. Next the gate complex proceeds to release both of the product molecules. In this design, binding of the translator releases the first product of the reaction products: A_n strand (`<ta^ an>`). The second product, B (`<tb^ b>`), is released upon binding of a *Fork helper* strand: `<b ta^>`. Lastly, the Fork_{AB} gate is sealed upon binding of the *Fork seal* strand `<i tb^>`, and as a result another seal waste molecule `<i>` is produced. This combination of Join and Fork gates consumes



(a) Signal integration: $\mathrm{Fsi}_n + \mathrm{A}_n \rightleftharpoons \mathrm{A}_n + \mathrm{B}$

(b) Activation function: $\mathrm{B} + \mathrm{Faf} \rightleftharpoons \mathrm{Faf} + \mathrm{E}$

(c) Weight accumulation: $\mathrm{A}_n + \mathrm{E} \rightleftharpoons \mathrm{E} + \mathrm{H}_n$

(d) Signal modulation: $\mathrm{A}_n + \mathrm{H}_n \rightleftharpoons \mathrm{H}_n + \mathrm{B}$

Figure 45: The correspondence of each of the reactions in a chemical neuron to the DNA-strand displacement interactions. Each reaction in the chemical reaction network (see table 6), is implemented as a combination of two-domain Join and Fork gates. For each reaction, a Join gate binds the two reactants in sequence, first displacing a waste molecule, and secondly displacing a translator molecule. In turn this translator molecule binds to the Fork gate which releases strands representing the products of the catalytic reaction. The translator displaces the first product, and then a Fork helper displaces the second product. Both Join and Fork gates are *sealed* upon binding of an appropriate auxiliary strand (labelled Join seal and Fork seal), which displaces the final incumbent bound <i> strand. See section 2.6.3 for an in-depth explanation of the underlying strand displacement mechanisms in the Join and Fork gate motifs.

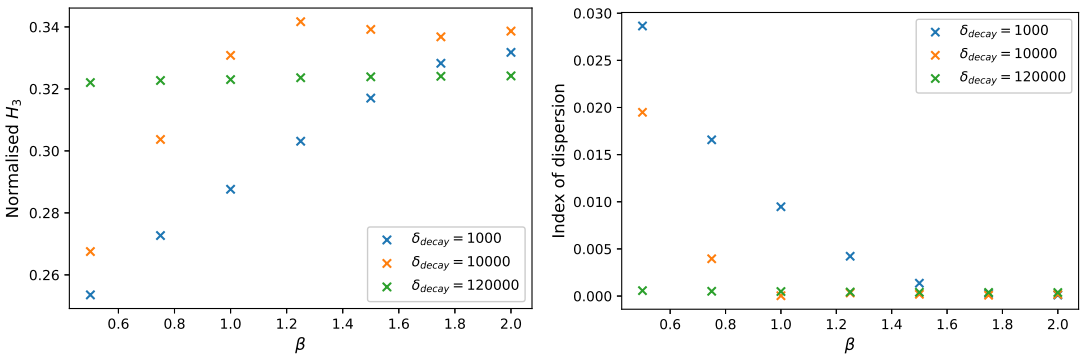
1 molecule of each reactant, and produces 1 molecule each of the products, ensuring equivalent stoichiometry to the abstract reaction.

5.3.1 Simulating the DNA neuron

The DNA neuron is simulated using Microsoft Visual DSD software [1] (see Appendix F). The timescales used in the all of the experiments are realistic, in a sense that the system could be synthesised and simulated in a laboratory in real time. The strand displacement reaction rates proposed in this model (see table 8) are within the realistic range that has been measured experimentally (Zhang and Winfree 2009). In order to reproduce the desired dynamical behaviours, the binding rates associated with the `ta`, `th` and `te` toeholds have been set to lower values than the other toeholds. Accordingly, shorter toehold and less reactive sequences are recommended for these domains (see table 9).

Each simulation starts with 10000 μM of gate complexes and helper strands needed for the computation by both Join and Fork gates in all catalytic reactions. The fuel molecules Fsi_n and Faf are initiated in the same way with the concentrations of 10000 μM. Lastly, the bolus size, or the amount of A_n species injected to the system at each spike, is set to $\beta = 1\ \mu M$. In order to implement depletion of H_n and E species garbage collection molecules are introduced: `{te^*}[e]` and `{th^*}[hn]`, which sequester and inactivate E and H_n respectively. These molecules are injected into the system periodically every 1000s with concentrations of 12 μM and 0.1 μM. This setup provides for a sufficiently stable environment, and allows for reproducible results. Notably, less frequent injections (e.g. $\delta_{\text{decay}} = 120000$) produce less stable behaviours, see fig. 46.

[1] The details of the software implementation are available on https://github.com/jf330/DNA-neuron/



(a) Normalised H_3.

(b) Index of dispersion.

Figure 46: The performance of different strategies of supplying garbage collector molecules to the system as a function of bolus size β. Here, I varied the temporal distance between subsequent injections of these species (δ_{decay}). The diversity of the weight set is measured using index of dispersion, i.e. the standard deviation divided by the mean of the weights. As expected, the more frequent but smaller injections (more similar to a decay with a constant rate) are more conducive to learning. The extreme case of $\delta_{\text{decay}} = 120000$ demonstrates that the system fails to learn if the garbage collection complexes are provided only once at the beginning of the simulation.

5.3.2 Increasing the number of input channels

In order to extend the system to accommodate for additional input channels the user needs to define a single new toehold domain `tiwan`, which is responsible for the weight accumulation in each of the N channels. Moreover, there are nine toehold domains that remain the same regardless of the number of input channels (`ta`, `th`, `tb`, `te`, `tfsi`, `tfaf`, `tism`, `tisin`, and `tiaf`). This means that the system with $N = 3$ input channels requires 12 toehold definitions ($9 + N$). The recognition of the inputs, as well as other two-domain strands in the system, is based on the long domains which are different for each channel. There are four long domains which remain the same regardless of number of channels (`b`, `e`, `faf`, `i`), and three which need to be defined when adding a new input channel (`an`, `hn`, `fsin`). Therefore, the system with $N = 3$ input channels requires 13 long domain definitions ($4 + 3N$). Notably, due to the different length requirements for long and short domains, it is easier to generate new long domains which are compatible with the reactivity requirements.

5.4 Learning

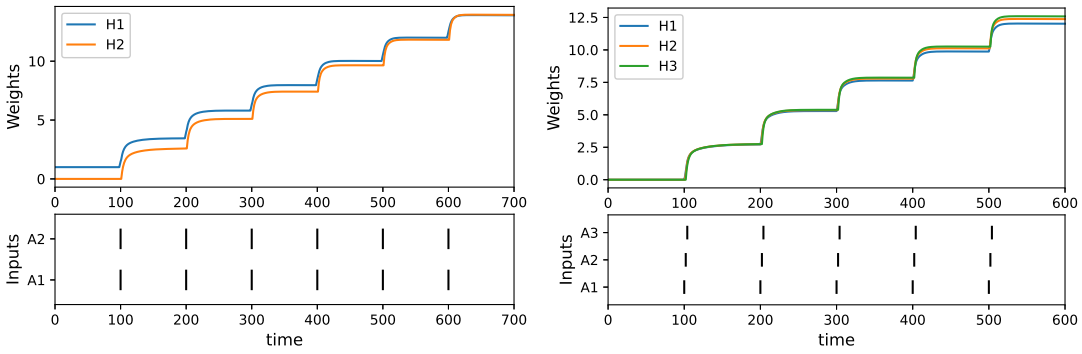
In order to determine whether the DSD implementation of a spiking neuron is capable of learning, a number of simulations were carried out using Visual DSD. Through these experiments I will demonstrate that both *infinite* and *detailed* compilation modes (see section 2.6.3) can produce the intended dynamical behaviours. Similarly, these behaviours can be reproduced in both simulations at low copy numbers (using Gillespie's stochastic simulation algorithm) and in the fluid limit (deterministic rate equations). In the following section, I will focus on the results from simulations carried out in infinite deterministic mode. Other simulations, including the ones using the detailed mode of Visual DSD, can be found in Appendix D.

5.4.1 Associative learning

I will first demonstrate that the system is capable of associative learning (fig. 47a). In this task, the neuron has $N = 2$ input channels, one with a low weight ($H_1 = 0\ \mu M$) and the other with a higher weight ($H_2 = 1\ \mu M$). The coincidence of low-weight input (A_1) with one that has a higher weight (A_2) allows for a quicker weight accumulation of the second channel. This is because the higher concentration of H_2 slows the progress of the weight accumulation reaction $\mathrm{A}_n + \mathrm{E} \rightleftharpoons \mathrm{E} + \mathrm{H}_n$. After just a few repetitions of the inputs in close temporal proximity, the weights associated with both channels converge. Initially the system would respond with a weaker activation (i.e. less E produced) if presented with input 1, but after conditioning, it would respond to both inputs with equal activation.

5.4.2 Temporal learning

Next, I will demonstrate that the system can be extended to an arbitrary number of input channels. This experiment shows that in this case the DNA neuron can learn to distinguish the temporal order of inputs by adjusting its weights. Figure 47b shows,



(a) Associative learning.

(b) Temporal learning.

Figure 47: (a) Example of associative learning in the DNA neuron. The weight associated with inputs from the first channel H_1 initially do not provide sufficient activation to the internal state B. The signal from A_1 channel is presented multiple times in coincidence with the signal from A_2. As a result the amount of H_1 molecules rapidly increases and after a couple of coincidences, the weights associated with both channels become equal. Thus, the system's reaction to the signal from conditioned (A_1) and unconditioned (A_2) channels becomes the same. (b) Example of Hebbian learning with three temporally correlated inputs. Here, the inputs from A_1 channel were injected first, and followed by inputs from A_2 and A_3 at fixed predefined temporal distance. After each pattern presentation, the weights increase, however not uniformly. The neuron learns to distinguish inputs by associating different weights depending on the order of their appearance. Here, a constant decay of E and H_n species is assumed with the rate constants of $k_E = 0.1$ and $k_H = 0.00002$ respectively.

for the case of 3 neurons, that the inputs which occur in closer temporal proximity to the moment when the learning signal molecules E (<te^ e>) become available, accumulate greater amounts of H molecules corresponding to their channels. Thus, the weights of inputs that spike in a synchronised manner are promoted. Moreover, the inputs which appear later in a sequence accumulate higher weights. This experiment qualitatively reproduces the effect of the STDP function on a spiking neuron's weights.

5.4.3 Bias detection in noisy input streams

I will now explore the system's performance on more complicated tasks, in which the neuron is presented with randomly generated streams of input spikes with a certain

statistical bias. More precisely, I will test the ability of the DNA neuron to detect frequency biases and temporal correlations. In each simulation, the neuron is exposed to 120000 seconds of inputs with a given type of statistical bias. The first 100000 seconds will be considered as a *learning phase*. After that time, the weights reach a homeostatic state, and remain on a steady level. The statistics of the weights are then measured over the remaining 20000 seconds.

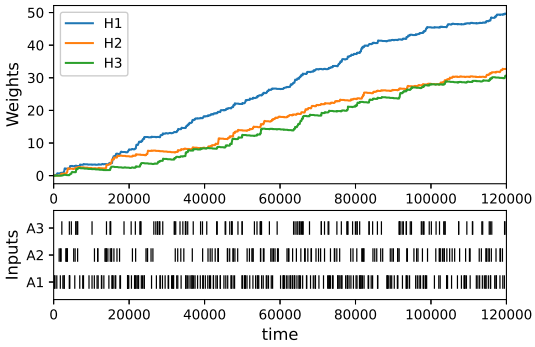
Frequency bias detection

Firstly, I will consider the frequency bias detection task (FB), as described before in section 4.4.2. In this task, the neuron is provided randomly generated boli to each of its N input channels. The waiting time between two successive boli of A_n is randomly distributed according to an exponential distribution with parameter $1/f_n$, where f_n is the frequency of the input boli to channel n. The system should then detect the difference in frequencies f_n between input channels. The task is considered as solved if (after a transient period) the ordering of the abundances of weights reflects the input frequencies, i.e. the number of H_i is higher than the number of H_j if $f_i > f_j$.

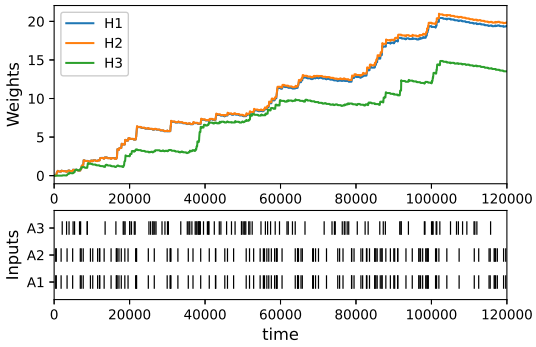
The neuron has three synaptic channels with randomly arriving inputs of differing frequencies (fig. 48a). The first channel is assumed to spike at a frequency of 0.002 Hz whereas the two remaining channels fire at a frequency of 0.001 Hz. The system is initialised with $H_n = 0$ for all three channels. As expected, the weights of the high-frequency input (H_1) increase faster compared to the other low-frequency inputs. This means that the DNA neuron can work as a frequency detector.

Temporal correlation detection

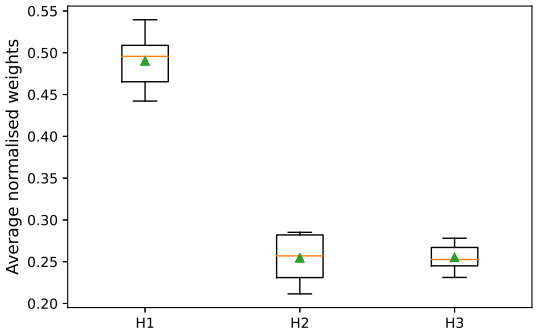
A more challenging task, which could not be solved by a single rate-coded neuron, is the learning of temporal correlations (see section 4.4.2). In this task, the input frequencies of each channel are the same, i.e. $f_i = f_j$ for all $i, j \leq N$, and set to 0.002 Hz. Pairs of channels i, j were chosen and time windows $\Delta\tau$ such that the probability to



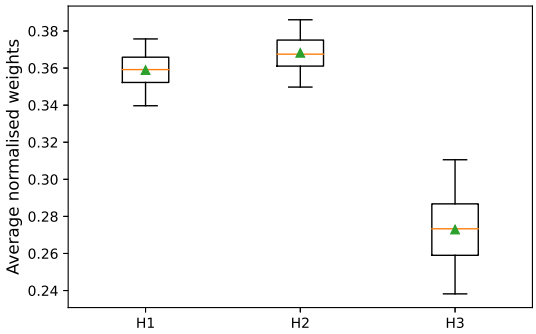
(a) Example of frequency bias learning.



(b) Example of temporal correlation learning.



(c) Average weights for frequency bias.



(d) Average weights for temporal correlation.

Figure 48: Examples of learning episodes for (a) frequency bias task, and (b) temporal correlation task. Normalised weights in the steady state for input data with (c) frequency bias, and (d) temporal correlation. The values presented here are averages over 20000s of stimulation, which were collected once the system reached a homeostatic state after 100000s. The experiment is then repeated 10 times. It can be seen from the figure that the steady state abundances of channel specific H_n molecules reflect the temporal order of the inputs provided. Moreover, the input channel that spiked in an asynchronous way (A_3) accumulated lower weights than the temporally correlated channels. In the frequency bias experiment, A_1 is assumed to come at frequency twice as high as that of the two other input channels. The DNA neuron achieves learning by accumulating steady state abundances of weight molecules H_n for different channels. At the end of the training phase, the weights of the channels with a greater probability of spiking will be high, while the channels with a lower frequency will reach a similar low level.

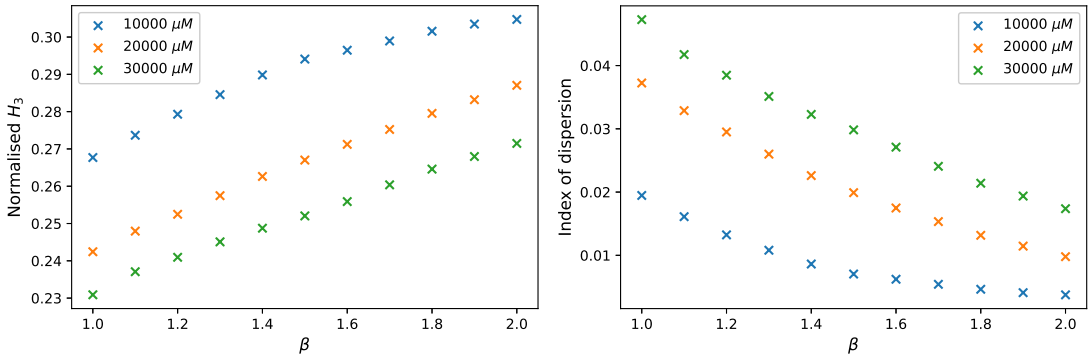
observe a bolus of j between time t_0 and $t_0 + \Delta\tau$ — where t_0 is the time where input A_i fired — is greater than the probability to observe a bolus between $t_0 - \Delta\tau$ and time t_0. In practice, such correlations are implemented as follows: If A_1 and A_2 are temporally correlated then each bolus of A_1 is always followed by a bolus of A_2 after a randomly generated but fixed time period δ. This means that the pair of correlated input signals always appears within a predefined temporal window, which is randomly generated for each trial.

The neuron is capable of solving the temporal correlation detection task in the sense that after a transient period, the weights indicate which channels are correlated and the temporal order implied by the correlation, i.e. if A_i tends to precede A_j, then the abundance of weight H_i should be higher than the abundance of H_j. Furthermore, if A_i is correlated with some other channel k, but A_j is not, then the abundance of H_i must be greater than that of H_j (see fig. 48d).

Next, I will investigate the ability of the systems to distinguish temporarily correlated inputs as a function of the bolus size β. To this end, I will use index of dispersion, i.e. the standard deviation divided by the mean of the weights, as a measure of diversity in the weight set. Figure 49 shows that the increase of β results in less diverse weight representations. As the amount of A_n molecules injected at each spike increases, the system's performance declines as a result of resource starvation. Each input spike results in a complete release of E molecules, regardless of the abundance of B molecules. Therefore, the weight updates become increasingly dependent on the frequency of inputs, rather than the temporal correlations embedded within the input data. This results in the steady state weight of the uncorrelated input (H_3) approaching the weights of the two other inputs.

Moreover, I tested different settings for the abundance of gate fuel molecules available at the beginning of each simulation. As the amount of available gate complexes increases, the ability of the system to distinguish temporally correlated inputs also increases. This shows that the performance of temporal correlation detection can be

increased at the cost of more fuel molecules, and thus longer simulation time.



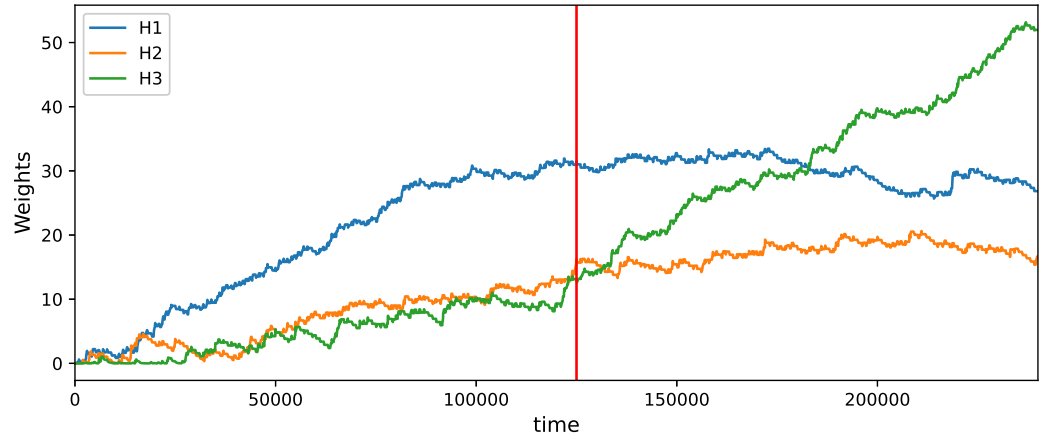
(a) Normalised H_3.

(b) Index of dispersion.

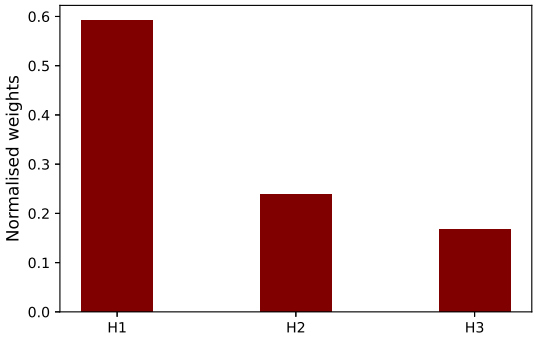
Figure 49: (a) The normalised value of the uncorrelated weight H_3 in a TC 2 task, and (b) index of dispersion for the set of weights at the steady state as a function of the bolus size β. The index of dispersion is a measure of diversity of the steady state weights, thus it indicates how well the neuron is able to distinguish between input channels. This means that an input stream with no bias would result in an index of dispersion ~ 0.

One particularly useful feature of the DNA neuron model, which was not present in any of the previous learning models in the DSD, such as Qian, Winfree and Bruck (2011) or Genot, Fujii and Rondelez (2013), is that it can readjust its weights based on the new learning episodes. Figures 50 and 51 show examples of such simulations. Here, the neuron is first exposed to one type of bias, which subsequently changes after $t = 120000$. In the case of frequency bias detection, firstly A_1 comes at a higher frequency than all other channels. Next, the bias embedded in the input stream changes and the inputs from A_3 become statistically overrepresented. After the next 120000s, the neuron's weights clearly reflect the new bias in the input data. Similarly, in the case of learning temporal correlation, I initially enforce a temporal correlation between inputs A_1 and A_2, where A_2 comes after A_1 with $\delta = 1$. Next, at $t = 120000$ the bias is changed, and a different temporal correlation is presented to the neuron: now the inputs from A_1 arrive at a fixed temporal interval after the inputs from A_3. In both cases, the weight distribution shifts in order to reflect the new kind of bias embedded in the input

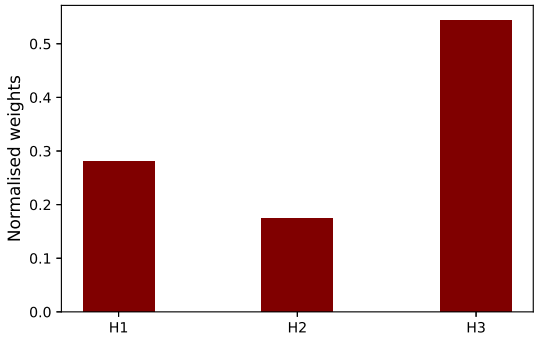
data.



(a) Example of frequency bias learning with input statistics changing during the simulation

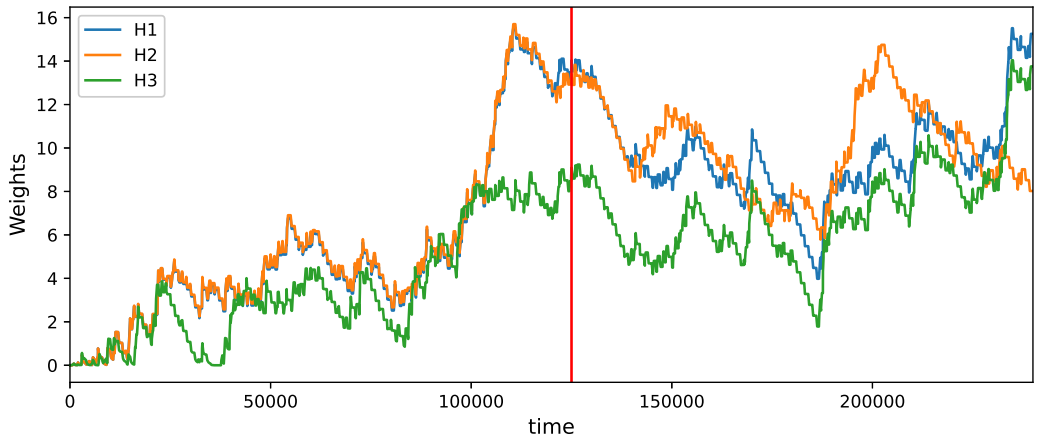


(b) Normalised weights at t=120000.

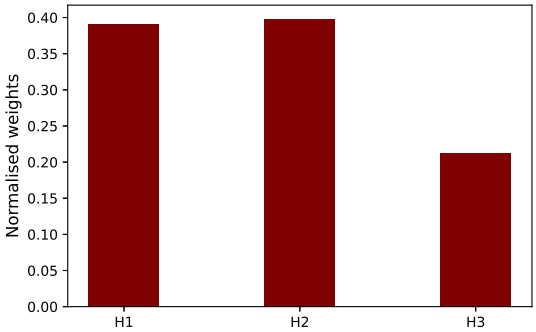


(c) Normalised weights at t=240000.

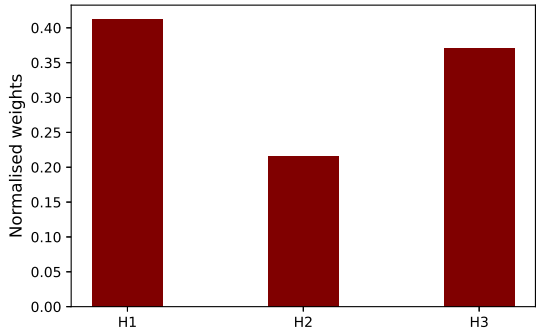
Figure 50: Example of a simulation where the statistical bias of the inputs changes during the trial. The simulation starts with inputs having a statistical bias towards inputs from A_1, which arrive twice as frequently as the other channels. At t=120000 the statistics change and A_3 becomes the statistically over-represented input channel. The neuron's weights reflect this change and the weight associated with the new biased input channel increases.



(a) Example of temporal correlation learning with input statistics changing during the simulation



(b) Normalised weights at t=120000.



(c) Normalised weights at t=240000.

Figure 51: Example of a simulation where the statistical bias of the inputs changes during the trial. The simulation starts with inputs having a temporal correlation between inputs A_1 and A_2, where the inputs from A_2 always arrive 1s after that of A_1. After 120000s, the weight associated with the second input channel H_2 is slightly higher than H_1, and both of them are significantly higher than H_3. At t=120000 the statistics change and a correlation between A_1 and A_3 is introduced, where the inputs from A_1 always arrive 1s after that of A_3. The neuron's weights reflect this change at t=240000. Now, the weight associated with the first input channel H_1 is slightly higher than H_3, and both of them are significantly higher than H_2.

5.5 Novelty detection

The DNA neuron is able to learn statistical relationships between a number of input channels. Crucially, this model is capable of readjusting its weights, and thus re-learning these relationships when they change. One application of this is to use it as a *novelty detector*, i.e. as a device that indicates that the statistics of the input has changed. Crucially, a useful novelty detector should be as much as possible, insensitive to the specific states of the system and only indicate changes of the state. To illustrate this, consider the following example: If the system has been in state $\mathbb{A}$ for a long time, then the novelty detector should be in its base state $\mathbb{0}$. If then, at some time $t = T_1$ the system changes its state to $\mathbb{B}$, then this should lead to the detector entering the state $\mathbb{1}$ transiently, before relaxing back to its base state $\mathbb{0}$. If now, the system changes again to a new state $\mathbb{C}$, then again the detector should indicate this by going into state $\mathbb{1}$ transiently.

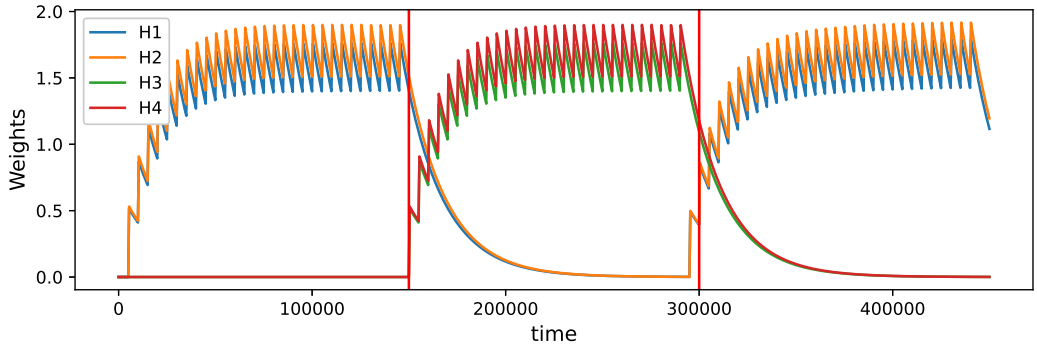
The basic idea of the novelty detector is that it indicates changes in the abundance of B. The amount of B molecules in the system is maximal when the weights are optimally adapted to the external inputs. Consequently, a sudden change of the external input will lead to a drop of the concentration of B. However, in the context of DNA computing, B itself cannot be used as an indicator, because it plays a crucial role in the functioning of the neuron. Using it in a different set of reactions would lead to interference. Hence, the novelty detector proposed here uses a proxy for B.

The input to the novelty detector is the two-domain strand `<b tfaf^>` (AF_{w1}). This is a waste molecule produced as a side-product of the activation function reaction depicted in Figure 45b. The production rate of AF_{w1} is therefore by construction exactly the same as that of the state species B. When B becomes available they interact with the $Join_{BFaf}$ gate, and in turn the exact same amount of AF_{w1} is released. Since AF_{w1} is a waste molecule which does not serve any purpose in the system, it can be used to facilitate additional functions, such as the novelty detection. To this end, an additional double-stranded DNA complex is introduced to the system: `[b]{tfaf^*}`.

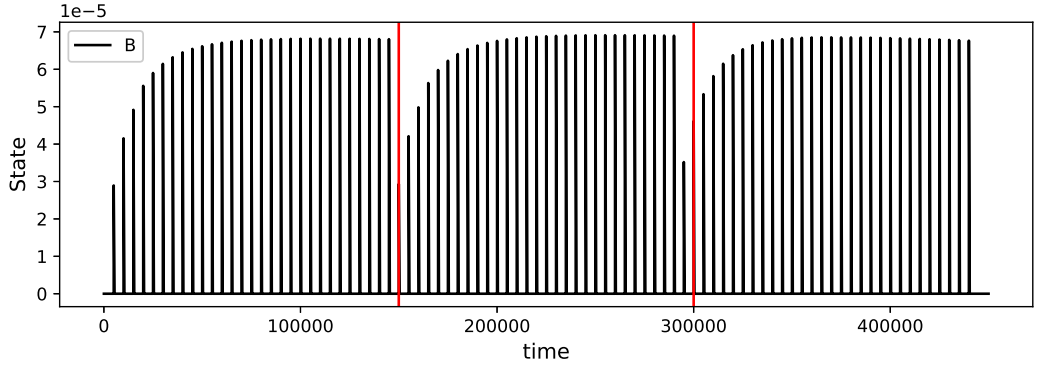
This *novelty reporter* species R_a allows for the binding of AF_{w1} species.

I have previously demonstrated that the amount of B reaches the maximum value, when the weights settle at the steady state. Knowing this value, from here on referred to as $[B]_c$, the same amount of the novelty reporter species is periodically injected into the environment. This naturally requires pre-established knowledge about the nature of the patterns presented to the neuron, which begs the question about the autonomy of this approach. One way of solving this problem is to run the program twice. First, in order to gather the statistics about the inputs, i.e. wait until the weights reach a homeostatic state and measure the amount of B produced in response to the inputs ($[B]_c$). Second time, in order to detect any further changes to the input statistics, i.e. periodically inject $[B]_c$ of reporter molecules to detect when the statistics of the inputs change.

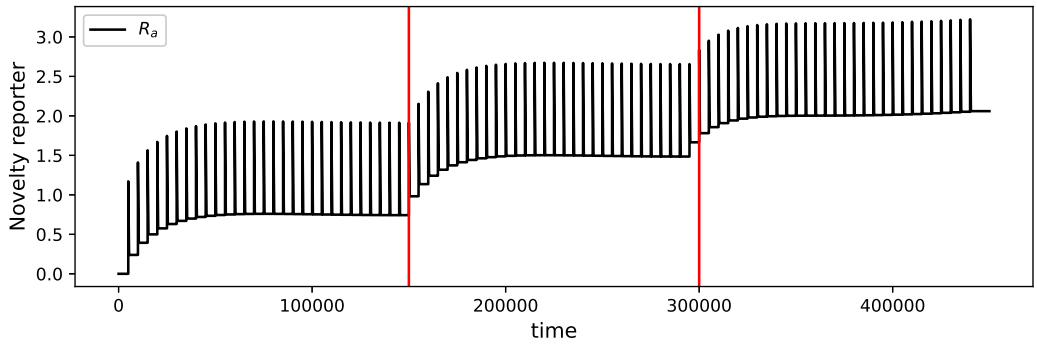
The simulation starts with a temporally correlated inputs from A_1 and A_2, where the inputs from A_2 follow the ones from A_1 in a close temporal proximity. Initially, as the simulation begins, the amount of B produced is far lower than $[B]_c$, and thus a small amount of novelty reporters remain in the system. As the neuron learns and the weights corresponding to the currently presented pattern reach the homeostatic state, the discrepancy between the concentrations of B and the novelty reporter decreases. Therefore, the amount of novelty reporter molecules stabilises, see fig. 52c.



(a) Synaptic weights - H_n



(b) State of the neuron - B



(c) Novelty reporter - R_a

Figure 52: Novelty detection in a DNA neuron - Approach A. Here, the neuron is presented with two types of temporally correlated patterns. After every 150000s the pattern being presented changes. Even though the frequency of inputs remains the same, the inputs which spike differ. This results in a decreased neuronal activation, and different set of weights being promoted in the process of learning. Here, I demonstrate a strategy for detecting such changes in the input stream. The number of times a change in the input pattern has occurred can be deduced by observing the concentration of *novelty detector* species.

At $t = 150000$, the pattern presented to the neuron is changed to input from A_3 proceeding the one from A_4. Since the weights associated with the inputs constituting this new pattern are low, the amount of B produced at each pattern presentation is also lower than before. This results in a surplus of novelty reporter molecules, which stabilise at a new higher level. This gives an indication of every subsequent presentation of novel inputs to the neuron. Notably, after switching the patterns again, at $t = 300000$ the same behaviour repeats. Initially, the amount of B released in response to the unfamiliar pattern is low. As the stimulation continues and the weights increase, the amount of B stabilises at $[B]_c$. On the contrary, the concentration of novelty detectors increases again, and finds a new equilibrium. The results of this mechanism are read out by measuring the average concentration of the reporter molecules, each increase in the concentration of R_a indicates that the inputs currently presented to the neuron are novel.

The second approach to novelty detection in a DNA neuron is based on a two-domain DSD seesaw motif (see section 2.6.3). Similarly to the previous example, AF_{w1} molecules are employed for this task. Firstly, a translator complex is used to exchange the <b tfafˆ> strand for another one with a different toehold domain which can be chosen arbitrarily. This allows to customise the properties of the novelty detector. To this end, a double stranded complex is added to the environment at the beginning of the simulation: <t1ˆ>[b]{tfafˆ*} in a quantity of 500 μM. This complex allows for binding of the AF_{w1} species and in response releases another two-domain single stranded species: <t1ˆ b > (S).

This approach to novelty detection is based on the well-known seesaw gate motif (Qian and Winfree 2011). The S molecules are used as an input species for the seesaw gate. Additionally, S decays with a rate of σ, such that S tends to a steady state determined by the abundance of B.

In order to implement the seesaw motif another type of double-stranded gate complexes is added to the system in a quantity of 5 μM. This seesaw gate complex

($Seesaw_{SR}$) has the following structure: `{t1^*}[b t2^]`, and allows for binding of one two-domain strand. When S strands bind to the gate complex, another two-domain species is released: `<b t2^>`, which will serve as a novelty reporter and will be henceforth referred to as R_b.

Once the gate has produced an R molecule, its binding properties change and it is able to consume R and release S. In that sense, the seesaw gate is reversible. As a result, the input species of the seesaw gate S and the output species R_b are in a constant competition for binding to the seesaw gate complex and a dynamic equilibrium is found between the concentrations of R and S. Their binding properties, and thus the equilibrium value, is defined by the nucleotide sequences of the toehold domains `t1^` and `t2^` but also their abundances. Indirectly, this equilibrium also depends on the concentration of B, which in turn changes the concentration of S and thus changes the balance.

In the process of learning, the DNA neuron accumulates the weights and therefore produces more B in response to the stimulus. When a novel pattern is being presented, the production of B molecules temporarily decreases, as a result of the weights associated with this input being low. As a result, the corresponding quantity of AF_{w1}, and therefore S also decreases. This means that the equilibrium between the S and R_b needs to change, as at that point the balance of their concentrations shifts. Therefore, the novelty is read out by periodically measuring the concentration of R_b reporter molecules. A drop in the average concentration of the molecular species R_b denotes a sudden decrease in the production of molecular species B (see fig. 53), and thus indicates that statistically novel input is being received by the neuron.

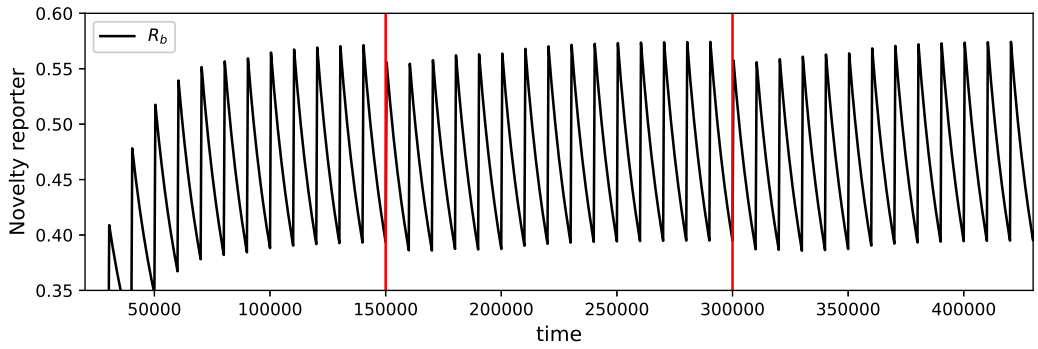


Figure 53: Novelty detection in a DNA neuron - Approach B. Here, the novelty is indicated by a dip in the level of R_b. The inputs, and therefore the weights and the state of the neuron remain the same as in fig. 52.

5.6 Tuneable activation function

The DNA neuron can be extended to implement activation functions with different degrees of nonlinearity. This can be achieved by employing an extendible polymer which can consume B and E_m molecules, where m indicates the length of the polymer, i.e. the number of B molecules which need to bind to it before the learning signal species L is released. The other reactions - signal integration and signal modulation remain the same as in the previously discussed DNA neuron model. The reactions within this new version of the DNA neuron are tabulated in table 10 and the binding parameters and the nucleotide sequences are shown in tables 11 and 12. See Appendix G for details of the Visual DSD implementation.

The key difference in this version of the DNA neuron is a different design of the activation function. The basic form of the activation function with $m = 1$ takes the form: `{tb^*}[b te0^]:[b te1^]<b>`. This complex allows for binding of B molecules; this in turn exposes the `te0` toehold which allows for binding of E_0. When E_0 binds to the complex, it displaces a long domain `b` and releases the learning signal L, which in the case of $m = 1$ is represented by three-domain species: `<b te1^ b>`. This series of interactions implements an activation function with the minimal

Function	Reaction
Signal integration	$\mathrm{Fsi}_n + \mathrm{A}_n \rightleftharpoons \mathrm{A}_n + \mathrm{B}$
Weight accumulation	$\mathrm{A}_n + \mathrm{L} \rightleftharpoons \mathrm{L} + \mathrm{H}_n$
Signal modulation	$\mathrm{A}_n + \mathrm{H}_n \rightleftharpoons \mathrm{H}_n + \mathrm{B}$
Activation function	$\mathrm{B} + \mathrm{E}_0 \rightleftharpoons \mathrm{E}_1$
	...
	$\mathrm{B} + \mathrm{E}_{m-1} \rightleftharpoons \mathrm{L}$

Table 10: List of reactions in a single DNA neuron with a tuneable activation function implemented as a chemical reaction network.

Signal Species	Toehold	Bind	Unbind
A_n	`ta`	1	10
H_n	`th`	0.001	10
B	`tb`	1	10
E_m	`tem`	5	10
Fsi	`tfsi`	1	10
Faf	`tfaf`	1	10
Ism	`tism`	1	10
Iaf	`tiaf`	1	10
Isi	`tisi`	1	10
Iwa_n	`itwan`	1	10

Table 11: List of toehold domains and their respective binding and unbinding rates in μMs^{-1} for the simulations carried out in the infinite compilation mode for a DNA neuron with a tuneable activation function.

degree of nonlinearity. This means that a comparatively small amount of B molecules is needed to produce a learning signal. Therefore, the system does not promote temporal correlations in its inputs, and thus is more sensitive to the frequency bias.

The DNA neuron can be tuned to exhibit varied sensitivity to frequency biases and temporal correlations by controlling the degree of nonlinearity of the activation function. This can be achieved by extending the polymer with additional segment to accommodate for binding of more B and E_m molecules. More precisely, this can be done by

Strand	Sequence
<ta^ a1>	$GACA + CCCTAAACTTATCTAAACAT$
<ta^ a2>	$GACA + CCCATTTCAAATCAAAACTT$
<ta^ a3>	$GACA + CCCATTACTAATCAATTCAA$
<th^ h1>	$CTCAG + CCCTTTTCTAAACTAAACAA$
<th^ h2>	$CTCAG + CCCTTATCATATCAATACAA$
<th^ h3>	$CTCAG + CCCTTAACTTAACAAATCTA$
<tb^ b>	$ACTACAC + CCCAAAACAAAACAAAACAA$
<te0^ b>	$CATCG + CCCAAAACAAAACAAAACAA$
<te1^ b>	$TACCAA + CCCTTATCATATCAATACAA$
<te2^ b>	$GTCA + CCCTTATCATATCAATACAA$
<te3^ b>	$GCTA + CCCTTATCATATCAATACAA$
<te4^ b>	$TATTCC + CCCTTATCATATCAATACAA$
<te5^ b>	$CACACA + CCCTTATCATATCAATACAA$
<tfsi^ fsi1>	$ACCT + CCCTATTCAATTCAAATCAA$
<tfsi^ fsi2>	$ACCT + CCCTATACTATACAATACTA$
<tfsi^ fsi3>	$ACCT + CCCTAATCTAATCATAACTA$
<tisi^>	$TAGCCA$
<tism^>	$CCCT$
<tiwa1^>	$CTCAATC$
<tiwa2^>	$CCTACG$
<tiwa3^>	$TCTCCA$
<i>	$CCCTTTACATTACATAACAA$

Table 12: List of the two-domain DNA strands and their respective nucleotide sequences in the DNA neuron with tuneable activation function.

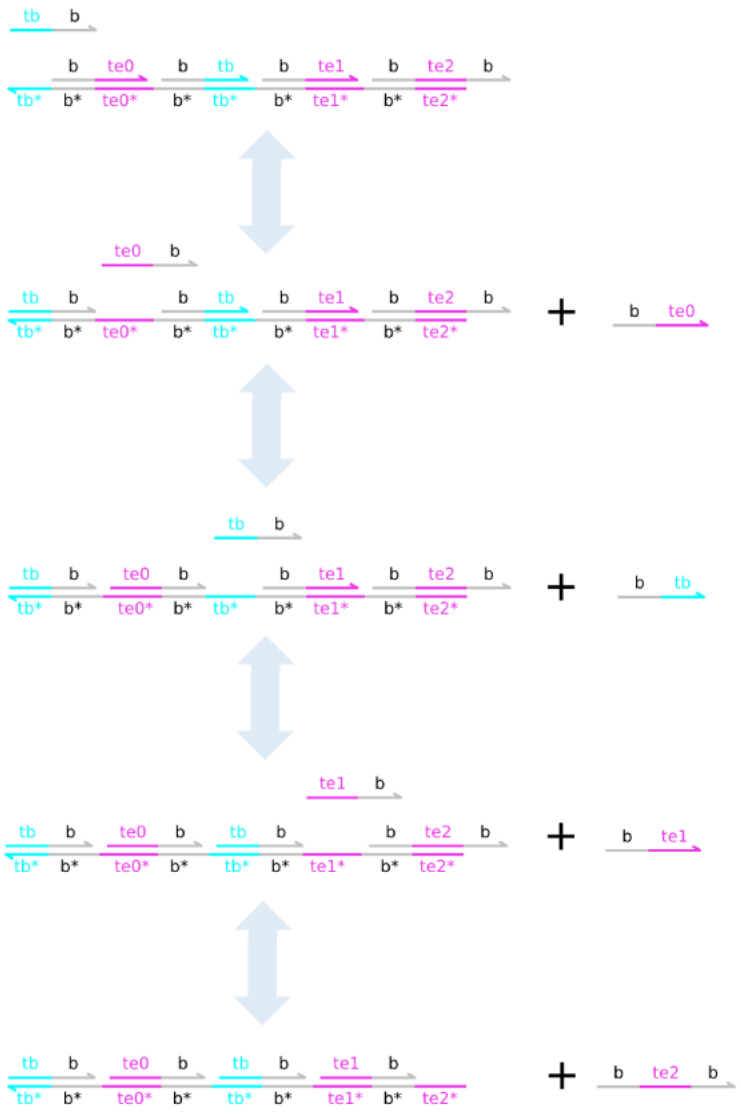


Figure 54: Diagram of the tuneable activation function for $m = 2$. The activation function is implemented by a long polymer which accommodates for binding of B and subsequent E molecules to its surface. These two species can bind to the polymer in an alternating manner. So first the binding of B frees up a `te0` toehold, next the binding of E_1 frees up `tb` toehold etc. This allows for an implementation of a repetitive extendible process which consumes B molecules. At the end of the process a three-domain learning signal molecule L is produced. In the case of $m = 2$, this molecule takes the form: `<b te2^ b>`. This mechanism is fully reversible, and thus once B molecules are removed from the system the polymer relaxes back to its original state.

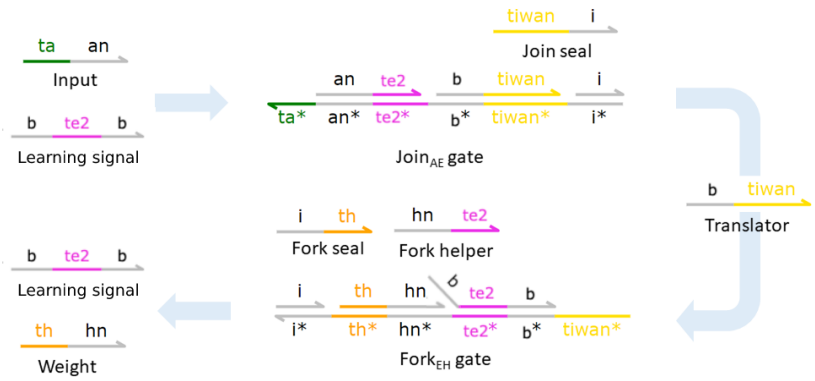


Figure 55: Weight accumulation ($m = 2$): $A_n + L \longrightarrow L + H_n$. The catalytic reaction implementing the weight accumulation function is facilitated by a Join gate and a modified variant of the Fork gate. In this variation, the first reactant of the Join gate and the first product of the Fork gate is now a three-domain species L. The initial Fork gate complex now has a long domain `b` branching out of the double-stranded structure. This modification is needed to ensure complementarity with the tuneable activation function.

adding subsequent segments `[b tb^]:[b tem^]` to the polymer, where m is the index of the next segment in the complex. This segments should be added before the last fragment which contains L species: `[b te1^]<b>`. This way, the activation function with $m = 1$ takes the form: `{tb^*}[b te0^]:[b te1^]<b>`, with $m = 2$: `{tb^*}[b te0^]:[b tb^]:[b te1^]:[b te2^]<b>`, etc.

The different form of activation function also requires a modified weight accumulation mechanism. In this variant, the first reactant of the Join gate and the first product of the Fork gate now becomes a three-domain species L. The initial form of the Fork gate complex now has a long domain `b` branching out of the double-stranded structure, and takes the following form: `[i]:[th^ hn]:<b>[te3^ b]{tiwan^*}`, see fig. 55. This modification is necessary in order to allow for L to catalyse the reaction.

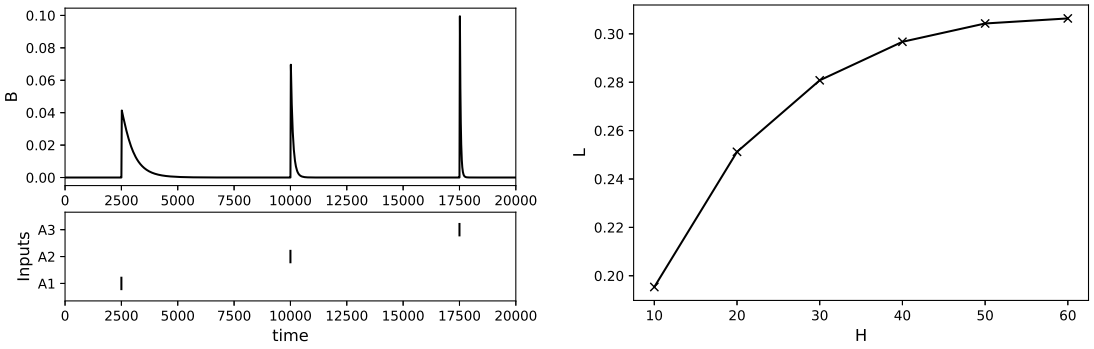
In this variation of the system, the simulation starts with different amounts of gate complexes and helper strands needed for the computation by both Join and Fork gates depending on their function. Signal modulation fuel molecules are initiated at 25000 μM, signal integration at 50000 μM, and weight accumulation at 10000 μM. The fuel molecules necessary for the signal integration mechanism: Fsi_n were initialised with

50000 μM. The simulation starts with a 10 μM of polymer molecules which facilitate the activation function and each of E_m molecules with 5 μM. Additionally 10 μM of molecules with a spatial orientation opposite to each of E_m are introduced. These two-domain strands: <tem^ b>, are necessary in order to maintain desirable dynamics of the activation function. Lastly, in all of the experiments the bolus size, i.e. the amount of A_n species injected to the system at each spike, is set to $\beta = 10\ \mu M$.

Another major difference in this version of the system is that E molecules (as well as its equivalent here - L) are not removed from the system. Instead, the new design requires the removal of B molecular species. In an effort to implement depletion of H_n and B species garbage collection molecules are introduced: {tb^*}[b] and {th^*}[hn], which allow for irreversible binding of these molecules.

Extending this version of the DNA neuron to accommodate additional input channels requires the user to define a single new toehold domain definition tiwan, which is responsible for weight accumulation in each of the N channels. Moreover, there are six toehold domains that remain the same regardless of the number of input channels (ta, th, tb, tfsi, tism, tisi). Therefore, the system with $N = 3$ input channels requires 9 toehold definitions $(6 + N)$. The recognition of the inputs strands, as well as all other two-domain strands in the system, is based on the long domains. There are two long domains which remain the same regardless of the number of channels (b, i), and three which need to be defined when adding another input channel (an, hn, fsin). Therefore, the system with $N = 3$ input channels requires 11 long domain definitions $(2 + 3N)$. Additionally, depending on the length of the polymer which facilitates the activation function there are at least two additional toehold domains required, i.e. for $m = 1$ te0 and te1. When the system is extended to accommodate for more non-linear activation functions it requires an addition of new toeholds domains: te2 for $m = 2$, te3 for $m = 3$, and so on.

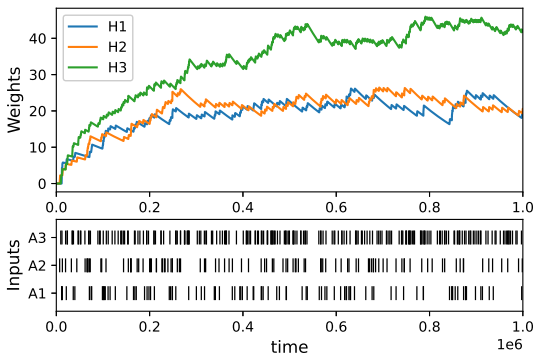
Figures 56 and 57 show that this version of the DNA neuron exhibits similar dynamics to the original system.



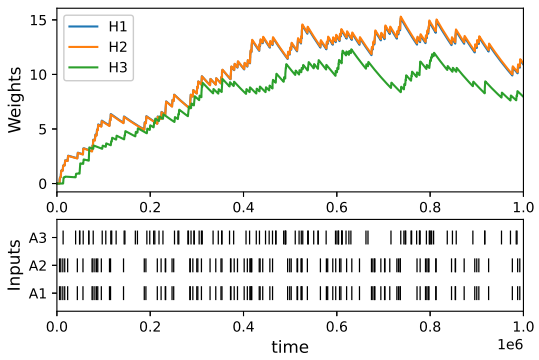
(a) Internal state (B) as a function of H_n.

(b) Learning signal (L) as a function of H_n.

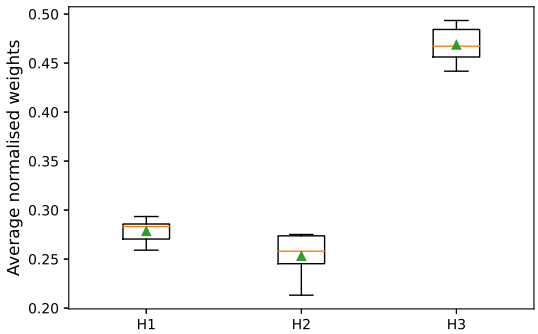
Figure 56: The internal state and activation of the DNA neuron with $m = 1$ as a function of differently weighted inputs (same as in fig. 44). (a) 10 μM of A_n molecules are injected to the system at $t = 100$, 300, and 500. Each channel has a different amount of H molecules associated with it: $H_1 = 10$, $H_2 = 30$, and $H_3 = 50$. This has the effect that the conversion of A_1 is slower than that of A_2, which is in turn slower than the conversion of A_3. (b) The influence of inputs on the concentration of L species as a function of its weight (H). The DNA neuron shows a different amount of activation given inputs from differently weighted channels. Additionally, a constant decay of H_n and B species is implemented with rate constants $k_H = 0.00002$ and $k_B = 2$.



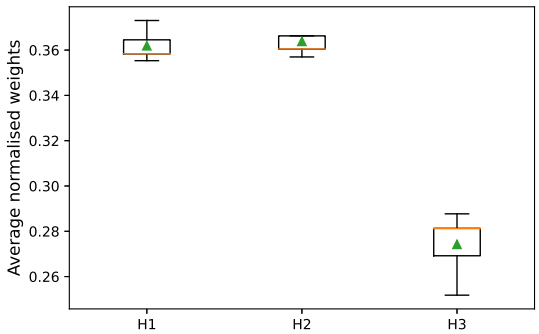
(a) Example of frequency bias learning ($m = 1$).



(b) Example of temporal correlation learning ($m = 3$).

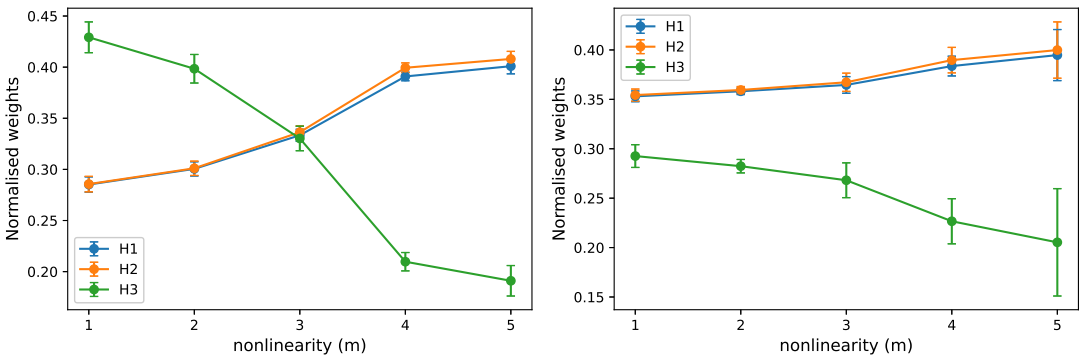


(c) Average weights for frequency bias ($m = 1$).



(d) Average weights for temporal correlation ($m = 3$).

Figure 57: Examples of learning episodes for (b) frequency bias task, and (a) temporal correlation task. Normalised weights in the steady state for input data with (d) frequency bias, and (c) temporal correlation. The values represent averages over 200000s of stimulation, and were only collected once the system reached a homeostatic state (after 800000s). We then repeat each experiment 10 times, and calculate the statistics. We define the temporal correlation as pairing of inputs from channels A_1 and A_2. The inputs from A_1 always precede that of the A_2 with a fixed temporal distance δ. We can see that the steady state abundances of channel specific H_n molecules reflect the temporal order of the inputs provided. Moreover, the input channel that spiked in an asynchronous way (A_3) accumulated lower weights than the temporally correlated channels. In the frequency bias experiment we assume A_1 to come at frequency twice as high as that of the two other input channels. The system learns by accumulating steady state abundances of weight molecules H_n for each channel. After training, the weights of the channels with a high input frequency will be high, whereas the ones less likely to spike are on a similar low level.



(a) FB + TC (b) TC

Figure 58: Normalised steady state weights as a function of the length of the activation function polymer. As the polymer is extended, the activation function becomes steeper and therefore requires a correlation of at least two signals to trigger learning. Therefore, the ability of the DNA neuron to recognise temporal correlations increases when m is high.

The ability of the DNA neuron to distinguish the two types of bias was tested on the task where A_2 is temporally correlated with A_1, and A_3 has a spiking frequency twice as high as the other input channels. In this experiment, A_2 always spikes 10s after A_1, i.e. $\delta = 10$. Both of the correlated channels spike with a frequency of 0.0001Hz, while the third input channel A_3 spikes with a frequency of 0.0002Hz.

Figure 58a shows the normalised steady state weights for 5 different systems with $m = 1, 2, 3, 4, 5$. The DNA neuron with a modified activation function is sensitive to frequency bias when the nonlinearity is low, i.e. $m = 1$. As a result, the weights corresponding to A_3 are high. In the case of high nonlinearity, the neuron promotes temporally correlated inputs, and thus the weights associated with A_1 and A_2 are higher. This shows that as the length of the polymer is increased, the neuron produces less activation in response to individual spikes and becomes more sensitive to the temporally correlated inputs. This is consistent with the previous findings discussed in Chapter 4.

Figure 58b shows the same graph, but for a modified experiment where temporal correlation between inputs A_1 and A_2 was enforced, and A_3 fires at the same rate as the other two input channels. Noticeably, the system still works (i.e. differentiates between

uncorrelated and correlated inputs) even for the lowest degree of nonlinearity in the activation function. The ability to distinguish the two types of signal further increases when $m > 1$.

Lastly, the DNA neuron was tested on the tasks previously performed by the CN and compartmentalised CN models (see figs. 31 and 42 respectively). Figure 59 shows that the DNA neuron with tuneable activation function indeed can solve a variety of TC and FB tasks to a high degree of accuracy. In order to better understand the relationship of the CN model proposed in Chapter 4 with the DNA neuron, the performance of both models was contrasted on a range of different tasks, see Appendix E.

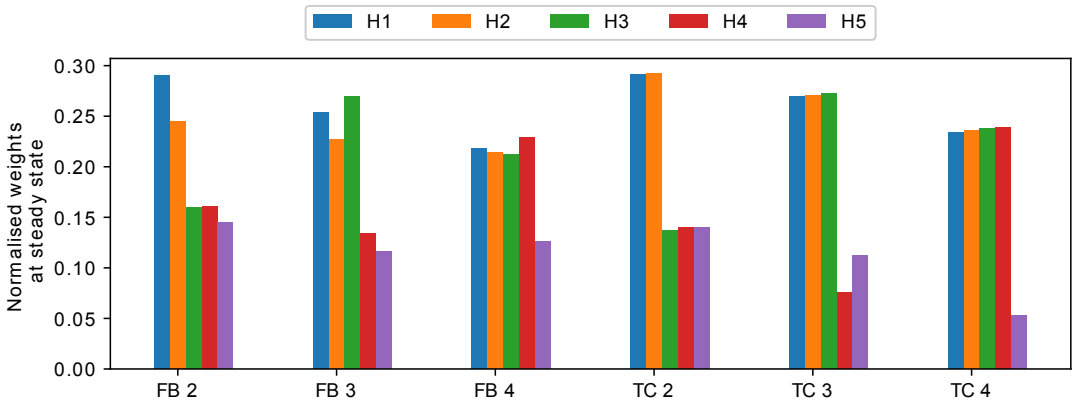


Figure 59: Normalised weights for a variety of TC and FB tasks. The first (blue) bar refers to the first weight, the second (orange) to the weight for the second channel and so on (same as fig. 31, but for the DNA neuron). Each value represents the average over 200000 time units of a single simulation. Data was only collected after the weights reached the steady state (after 800000 time units). The non-linearity was set to $m = 5$ for the TC, and $m = 1$ for the FB tasks.

5.7 Chapter summary

In this chapter, I proposed a design for the first DNA-based model capable of autonomous learning of spatio-temporal patterns. The system replicates the behaviour of a spiking neuron with STDP. Through extensive simulations, I have demonstrated that it can detect both frequency biases and temporal correlations. Other useful functions

of this approach include: scalability to an arbitrary number of input channels, being able to readjust its weights once the input statistics change, and novelty detection, i.e. indication of when a change in input statistics has occurred.

This model meets all necessary criteria for being considered autonomous (see section 4.1). This design is an implementation of the previously discussed chemical neuron (see Chapter 4), however it provides a concrete framework for successful implementation in a wet laboratory. This model inherits the key feature of the CN, in that it is autonomous. Both the learning algorithm and the neuronal functions are implemented within the model itself. The DNA neuron is built using a well established and experimentally proven design motif of two-domain DNA strand displacement (Cardelli 2010). This framework promises to provide building blocks for implementations which are efficient and free of cross-talk. Moreover, it has been used to recreate other molecular circuits described as chemical reaction networks (Chen et al. 2013), and therefore offers a plausible strategy for physical realisation in laboratory.

The advantage of employing DNA circuits over GRNs is that the former are much easier to program and synthesise, and they also have a much reduced propensity to engage in cross-talk. A disadvantage, however, is that the DNA circuits need to interface with the *in vivo* target systems. How to do this in a general way remains an open question, but there have been a number of previous systems that demonstrated how this can be done (Groves et al. 2015; Oesinghaus and Simmel 2019; Jung and Ellington 2014). One example of such interface can be the work of Douglas, Bachelet and Church (2012) who proposed a design for an autonomous DNA nanorobot capable of transporting molecular payloads to cells. This system is capable of detecting certain cues related to the cell surface and translate them into DNA-based signals.

The DNA neuron has a few crucial differences when compared to the CN (see Chapter 4). Unlike the CN, the basic DNA-based model can be built using four main catalytic reactions, each implemented using a combination of two-domain Join and Fork gates. Moreover, the DNA system requires several types of fuel molecules in order to drive

the reactions. The sealed variant of the gates was used in order to avoid rebinding of the products, and thus making the reactions unidirectional. A significant aspect of the neuron's biochemical implementation was a customisable activation function, which allows the user to control the degree of nonlinearity of the activation function. In most spiking neuron models, this would ideally be a step-function. In molecular systems, this cannot be realised directly. Usually, it is instead approximated by Hill enzyme kinetics. In section 5.6, I have demonstrated how more nonlinear activation functions can be implemented in the DNA-strand displacement. Extendible polymers which allow for several B molecules to bind to a single long DNA complex were shown to be able to approximate such Hill dynamics. This extension allows the DNA neuron to put more emphasis on learning of the temporal patterns, and ignore the ones which spike more frequently but in an uncorrelated manner.

Chapter 6

Conclusions

6.1 Thesis summary

This thesis aimed to investigate the possible directions towards modelling autonomous neuromorphic systems and find possible substrates for biological implementation.

In the first research chapter of this thesis, I have investigated the necessary components for multi-pattern classification in spiking neurons. I proposed an alternative neuronal model - GNM, which can perform the same task as the MST and vastly simplify the computation. The GNM proved to be on par with other neuron models, such as LIF and MST, when trained using the aggregate-label learning algorithm. I have also examined the possibility of implementing this neuronal model in a molecular setup by interpreting the continuous time differential equation model as a chemical reaction network. Lastly, I demonstrated how the synaptic weights can be encoded as reaction rate constants between certain molecular species, thus making the first step towards biochemically implementable neuromorphic systems.

The investigation from Chapter 3 led to the introduction of the chemical neuron: a network of chemical reactions which incorporates both learning and classification mechanisms in a single system. I proposed a biochemically consistent model for a

spiking neuron, and demonstrated that it is capable of performing simple tasks previously shown to be realisable in synthetic biology - associative learning, as well as more complex tasks including full Hebbian learning of spatio-temporal inputs. Additionally, an alternative version of the system was presented, which is constructed using established biochemical motifs. This model lends itself to an interpretation as an artificial single-celled organism.

In search for a suitable substrate for biochemical implementation of autonomous neuromorphic learning systems, I investigated the usefulness of DNA-based computation. The main contribution of Chapter 5 was the model of the DNA neuron, which is the first fully autonomous learning system built entirely with two-domain DNA-strand displacement reactions. Moreover, it is also the first model which allows to learn the statistics of spike-based inputs in DNA. Importantly, employing the framework of DNA-strand displacement allowed for constructing a system which could be realistically synthesised in a wet laboratory.

6.2 Discussion

Compared to the rate-coded neuron models, such as the perceptron, spiking neurons are more closely related to the realistic biological neuronal systems (Brette 2015). They are also proven to be computationally more powerful (Maass 1997). The spiking neurons distinguish signals based on their temporal arrangement, hence allow for more precise signal filtering and more energy efficient computation. These promising findings led to the development of a field centred around designing mathematical models which could closely mimic the functions exhibited by real neurons in the brain. However, it is still an open question whether all of these functions are relevant, and at what level of abstraction should these systems be analysed. As a result, over the years researchers have proposed multiple vastly different neuromorphic models of neuronal systems.

There are many different elements which may need to be taken into consideration

when designing a spiking neuron. These may include neuronal functions such as refractory period, post-spike membrane reset, or discrete spiking. Nevertheless, not all of the previously published neuromorphic models possess all of these features. For example, the Izhikevich neuron (Izhikevich 2003) closely mimics realistic dynamic behaviour of biological neurons and populations of such neurons can reproduce well-known firing patterns present in the brain. On the other hand, models like the multi-spike tempotron (Gütig 2016) offer a powerful framework for learning multi-class stimuli, however they are also internally complex and do not include a hard post-spike reset and refractoriness.

This observation raises the question of what are the essential elements which a biologically inspired neuron model needs to possess in order to be considered "neuromorphic". Some would argue that the models should be judged according to their "biological plausibility", since the word neuromorphic derives from the words *neuro* - related to the nervous system, and *morphic* - having the form or structure. However, this word typically refers to the systems which can be implemented on *neuromorphic hardware*. This special type of computational devices allow for more efficient simulation of spiking neurons, compared to standard digital computers. Importantly, some of the spiking neuron models, such as LIF, are more suitable for simulation on analog versions of such computational devices. On the other hand, more complex models, such as multi-spike tempotron, offer an improved potential for solving difficult computational tasks, such as multi-class learning in a single neuron. However, due to their internal complexity, they are computationally expensive and difficult to implement outside of digital computers.

A reasonable question to ask is whether this internal complexity of models like MST is indeed necessary. As shown in Chapter 3, the MST model can be vastly simplified, while still retaining its ability to compute. Importantly, the spikiness has been shown to be unnecessary (given that discrete threshold crossing events can still be registered by an outside observer). The GNM models which have the lowest degree of nonlinearity

(i.e. $\eta = 0$) were revealed to perform equally well or better than more complex GNM's with $\eta > 0$. This investigation has shown that the degree of temporal autocorrelation of the membrane potential is a key parameter for learning in aggregate-label fashion, and hysteretic post-spike reset of the membrane potential is redundant. This is an unexpected discovery, since the sensitivity to threshold and the post-spike reset are typically present in the spiking neuron models.

The notion of biological plausibility frequently appears in the neuromorphic literature. However, it is often ill-defined and vague. As pointed out before, different neuronal models offer certain subsets of the properties of the real neurons. One such example could be absolute refractory period, which completely disables the neuron from integrating the inputs for a period of time after an output spike was elicited. While it is useful in certain tasks involving larger neuronal networks, it also seems to damage the neurons ability to compute in multi-spike aggregate-label learning tasks (see fig. 16). The investigation conducted in this thesis has shown, that it is detrimental especially in the case when the neuron is supposed to release multiple spikes in a close temporal proximity. Even though the refractoriness has been observed in real neuronal circuits, it is unclear whether it serves a particular role or is a side effect of resource limitations in this particular implementation.

In the first research chapter of this thesis, after first establishing what are the necessary components for spiking neuron computation and designing the initial model for approximating neuronal models in chemical reaction networks, it became clear that there is a gap in understanding of neuronal systems on the abstract mathematical level and in terms of their actual biochemical implementation in the brain.

In an effort to replicate neuromorphic functions in synthetic biological systems, first it is necessary to understand all requirements for such implementation. A particular challenge in designing these systems, before they can be useful, turns out to be endowing them with *autonomy*. For the purposes of this thesis, I have established a list of necessary conditions for the system to be autonomous: (i) The learning needs

to be unsupervised, since providing feedback would require an external agent. (ii) The learning algorithm needs to be internal to the system itself. This means that the simulation cannot be stopped to perform weight adjustment. (iii) The storage of synaptic weights cannot rely on external storage. They need to be represented internally by the system. (iv) Signal modulation also needs to be implemented internally. There needs to be a mechanism in place which allows the weights to scale the impact of their respective inputs on the state of the neuron. Meeting all of these criteria is a difficult challenge, however I have demonstrated that a neuromorphic learning system which fulfils these requirements can be designed as a chemical reaction network. The one particularly troublesome requirement is the need for implementing the learning algorithm within the chemical reaction network. The supervised aggregate-label learning framework used to train the MST requires an entire history of pre-synaptic inputs in order to calculate the weight updates. This method necessitates keeping track of the eligibility traces for each of the input channels. Moreover, the simulation needs to be stopped after each trial in order to adjust the synaptic efficacies. Similarly for unsupervised learning, such as STDP, it is assumed that an external observer applies a learning algorithm when a neuron fires, and readjust the synaptic weights based on the history of pre-synaptic inputs. While this is trivial to implement in digital computers, it becomes much more difficult when implemented in biology.

In the biological neural circuits, the learning is attributed to the intracellular calcium signals, which are sensitive to the coincidence of pre and post-synaptic activity through the voltage dependence of synaptic N-methyl-D-aspartate (NMDA) receptors. It is well established that the induction of long-term synaptic changes requires these calcium signals to reach a specific plasticity induction thresholds. Gütig (2016) uses this biological mechanism to justify his correlation-based learning framework. His simplified version assumes selecting the 10% most eligible synapses to undergo depression or potentiation. On the other hand, this mechanism is often also attributed to the unsupervised STDP learning, where the synaptic weight update size is determined by the fraction

of NMDA receptors open at the time of input spike. This shows that while we have some understanding of the mechanism which underpin the learning in the real neuronal circuits, there is still some uncertainty in how is it implemented on the chemical level of abstraction.

Spiking neurons are capable of detecting two types of biases embedded within the input data streams. Firstly, they are specifically designed to detect temporal correlations. The temporal nature of integration kernels, allows these neurons to accumulate information about the incoming signals through time. Therefore, two inputs have a greater probability of activating the neuron when they occur in close temporal proximity. Secondly, the channels which are more likely to produce the input spikes compared to the other channels are also promoted. This is because they have a greater probability to coincide with the other inputs, and therefore cause activation. In principle, all of the tasks that the SNNs perform are in fact a combination of these two types of biases. Examining the neurons ability to detect these biases separately has shown that there exists an optimal degree of nonlinearity in activation function which allows for detecting them. The higher nonlinearity is needed for a better recognition of temporal correlations, while the frequency bias can be detected by a very simple models with no activation whatsoever (see section 4.2). This means that for biochemical models, such as CN, there is an optimal chemical composition for detecting different types of biases in the environment. The additional reactions, of course come at the cost of higher energy consumption (see fig. 39). Simpler models, while retaining some ability to recognise temporal correlations between inputs, lose the ability to provide more diverse sets of weights.

In order to synthesise such a system in a laboratory, it is first necessary to identify a suitable and reliable substrate for this type of computation. The DNA-strand displacement framework has emerged as a promising alternative to previously popular gene regulatory networks. This is mainly due to the fact that the simulation can closely approximate the behaviour of real DNA circuits. Another reason is the lack of

some problems common to the GRNs, such as cross-talk and non-specific binding between the biochemical molecules. The DNA neuron, which realises the simplest case of minimal nonlinearity, has been shown to be sufficient to realise both temporal and frequency bias learning. Additionally, the DNA neuron can be extended to accommodate for more nonlinear activation functions, and thus improve its ability to detect temporal correlations.

While the DNA neuron model is autonomous and minimally complex, it also lacks certain features which would make it easily deployable in biological contexts. Most notably, it assumes that the input signals are two-domain DNA strands. While interesting in itself, mechanisms allowing to interface with biological cues which are not in a form of DNA would be needed to fully explore the potential of this model. This aspect still remains an open question, but there have been a number of previous systems that indicate possible pathways (Groves et al. 2015; Oesinghaus and Simmel 2019; Jung and Ellington 2014). For example, Douglas, Bachelet and Church (2012) demonstrated that it is possible to build an autonomous DNA nanorobot capable of transporting molecular payloads to cells. The system is endowed with an ability to sense cell surface inputs for conditional activation. This kind of nanorobots can be loaded with a variety of materials, for example antibody fragments. This device is controlled by an aptamer-encoded logic gate, which allows it to respond to different environmental cues. Endowing such a system with additional computational capabilities beyond logic gates, for example detecting temporally correlated cues, could make it even more flexible and capable of reacting to more complex stimuli. The DNA neuron could in theory be used as a detection layer for this type of DNA-based devices.

Implementable learning system which could interact with arbitrary biochemical signals could be put into practical use in a range of useful tasks. One example of such a use case would be intelligent drug delivery based on changing conditions inside of the living organism. Theoretically, such a machine could be used to deliver medicine at precise times when the concentration of a certain chemical in the organism reaches a

threshold level. Other examples could be environmental sensing or automated production of medicine, both of which require a high degree of control and could benefit from autonomous detection of environmental cues.

6.3 Future work

This thesis demonstrated several novel contributions to neuronal modelling and building intelligent systems in the context of synthetic biology. Nevertheless, during this investigation several shortcomings as well as possible extensions were identified. These topics were considered outside of the scope of this thesis, and left for future researchers to investigate. In the following sections I am going to discuss some of these ideas.

6.3.1 Backpropagation for networks of GNM

In section 3.7.2, I have proposed a framework for multi-layered learning in the networks of GNMs. This approach was able to learn on par with other neuronal models and other training approaches. Nevertheless, the advantage of using a greater number of neurons wasn't clear enough (see fig. 19). In order to validate this approach it would be beneficial to conduct more in-depth analysis of the underlying algorithm and test it on more complex datasets, such as classification of MNIST handwritten digits (LeCun and Cortes 2010).

6.3.2 Building networks of CN

In this thesis, the research has mainly focused on the modelling of individual neurons in the context of synthetic biology. Nevertheless, building functional networks out of these chemical processing units remains a challenge. In particular, these systems are unsupervised, while deep neural networks are typically trained in a supervised way. Another issue could be an immediate and fixed amount of inputs which needs to be injected to the post-synaptic neurons in the next layer of the network. This feature

would need to be facilitated by a separate biochemical mechanism, which would allow for a controlled released of the precursor molecules I to the hidden units in the network.

6.3.3 Wet-lab implementation of DNA neuron

While the DNA neuron proposed in Chapter 5 promises a model which could realistically be synthesised in a laboratory, a number of caveats and potential challenges remain:

- The proposed system is a two-domain DSD design. It assumes that all species which facilitate computation, as well as the inputs are such two-domain DNA strands. The signals in the real world would not necessarily be DNA strands of this kind. Therefore, in order to perform its function, the DNA neuron requires an additional system which could interface with non-DNA cues in the environment. How to do this, however, remains an open question, and still has not been answered in other experimental work.

- In order to scale the system to accommodate for more input channels additional short domain sequences need to be defined for each new channel. Therefore, the scaling of the DNA neuron is limited by the availability of orthogonal short domain sequences. This suggests that there exists a theoretical hard limit for the number of channels which could be modelled. One solution to the limitation of toeholds could be a redesign based on localised design principles, i.e. physical separation introduced between the channels. A similar case of compartmentalisation has been shown to be feasible in DSD systems by Chatterjee et al. (2017).

- A particular limitation of the DNA computing is that the underlying nucleotide sequences may undergo unforeseen interactions. The toeholds may bind erroneously or form unintended complexes (Berleant et al. 2018), which may be difficult to predict and account for from the domain-level perspective. Therefore, it

would be useful to conduct a nucleotide sequence level analysis. This way, such interactions could be avoided, or at least the error rate could be estimated.

- Describing the model as a neuron encourages the question of building networks of these units. Such networks could carry out more complex computational tasks. A major obstacle in building networks of DNA neurons could be the immediate injection of A species to the post-synaptic neuron. This would necessitate a re-thinking of how the activation function is implemented. Nevertheless, the use of long polymers could be a potential answer to this challenge. Incorporating a buffered design of a long polymer, could allow for a programmed release of a certain number of input species, once the activation signal is produced (Lakin et al. 2012).

6.4 Publications

The list of publications by the author relevant to the thesis:

- Fil J., Dalchau N., Chu D.; Autonomous unsupervised learning for a spiking neuron implemented in DNA strand displacement reactions. 27th International Conference on DNA Computing and Molecular Programming. September 2021

- Fil J., Chu D.; Minimal Spiking Neuron for Solving Multilabel Classification Tasks. Neural Computation 2020; 32 (7): 1408–1429. July 2020

Appendix A

ϑ^* gradient

The second learning algorithm proposed by Gütig (2016), after the correlation based aggregate-label learning, requires the calculation of the gradient $\nabla_w \vartheta^*$ of threshold allowing for a certain neural response. This is done in two steps. After each iteration, when the neuron is exposed to the stimuli, the error is calculated. Firstly, in order to narrow down the range of possible answers (the recursive operations used in the second step can be computationally expensive), interval halving is used until an extra spike is produced, with upper bound being the threshold which gives $k-1$ spikes, and lower bounds corresponding to ϑ which results in k spikes. In the second step ϑ_k^* is determined numerically by finding the root of expression $[\vartheta - \nu_{\max}(\vartheta)]$, where $\nu_{\max}(\vartheta)$ corresponds to the maximum sub-threshold membrane potential value. The membrane potential equation for the multi-spike tempotron (Eq. 25) with the threshold set to a critical spike threshold ϑ^*, can be rewritten as:

$$V(t) = V_o(t) - \vartheta^* \sum_{t_j^f < t} \exp\left(-\frac{t - t_j^f}{\tau_m}\right) \tag{41}$$

where function V_o is equivalent to the voltage equation for the leaky integrate-and

fire neuron with no post-spike reset:

$$V_o(t) = \sum_{i=1}^{N} w_i \sum_{t_i^f < t} K(t - t_i^f) \tag{42}$$

It is assumed that for each ϑ^* there is a unique t^* such that:

$$\vartheta^* = V(t^*) = V_o(t^*) - \vartheta^* \sum_{f=1}^{m} \exp\left(-\frac{t^* - t_j^f}{\tau_m}\right)$$

where m denotes the number of output spikes produced before t^*, and $t_j^f < t^*$ for $f \in \{1, ..., m\}$.

Since the membrane potential is being reset every time it reaches the threshold, all output spike times t_{j}^f and t^* should satisfy:

$$\vartheta^* = V(t^*) = V(t_j^f)$$

Hence for each afferent $i \in \{1, ..., N\}$:

$$\vartheta_i^{*\prime} \equiv \frac{d}{dw_i}\vartheta^* = \frac{d}{dw_i}V(t^*) = \frac{d}{dw_i}V(t_j^f) \tag{43}$$

where $\vartheta_i^{*\prime}$ denotes the ith component of the desired gradient. The expression ϑ^* depends on the synaptic weights, and its derivative with respect to w_i is:

$$\vartheta_i^{*\prime} = \frac{\partial}{\partial w_i}V(t^*) + \sum_{f=1}^{m} \frac{\partial}{\partial t_j^f}V(t^*)\frac{d}{dw_i}t_j^f \tag{44}$$

Note that the last term in the derivative: $\frac{\partial}{\partial w_i}V(t^*)\frac{d}{dw_i}t^* = 0$ was dropped as $V(t^*)$ is a local maximum with $\partial V(t^*)/\partial t^* = 0$. Similarly, for every $k \in \{1, ..., m\}$:

$$\frac{d}{dw_i}t_{\mathrm{j}}^k = \frac{\partial}{\partial w_i}V(t_j^k) + \sum_{f=1}^{k} \frac{\partial}{\partial t_j^f}V(t_j^k)\frac{d}{dw_i}t_j^f \tag{45}$$

From which the following can be obtained:

$$\frac{d}{dw_i}t_j^k = \frac{1}{\dot{V}(t_j^k)}\left(\vartheta_i^{*\prime} - \frac{\partial}{\partial w_i}V(t_j^k) - \sum_{j=1}^{k-1}\frac{\partial}{\partial t_j^f}V(t_j^k)\frac{d}{dw_i}t_j^f\right) \tag{46}$$

where the time derivatives are:

$$\dot{V}\left(t_j^k\right) \equiv \frac{\partial}{\partial t}V(t)\bigg|_{t=t_j^k} \tag{47}$$

Eq. 44 can be solved for $\vartheta_i^{*\prime}$ by refactoring Eq. 46 into the following form:

$$\frac{d}{dw_i}t_j^k = \frac{1}{\dot{V}(t_j^k)}[\vartheta_i^* A_k + B_k] \tag{48}$$

Here A_k and B_k are recursive equations:

$$A_k = 1 - \sum_{f=1}^{k-1}\frac{A_j}{\dot{V}(t_j^f)}\frac{\partial}{\partial t_j^f}V(t_j^k) \tag{49}$$

and

$$B_k = -\frac{\partial}{\partial w_i}V(t_j^k) - \sum_{f=1}^{k-1}\frac{B_j}{\dot{V}(t_j^f)} - \frac{\partial}{\partial t_j^f}V(t_j^k) \tag{50}$$

At time t^*, these two can be reformulated as:

$$A_* = 1 - \sum_{j=1}^{m}\frac{A_j}{\dot{V}(t_j^f)}\frac{\partial}{\partial t_j^f}V(t^*) \tag{51}$$

and

$$B_* = -\frac{\partial}{\partial w_i}V(t^*) - \sum_{j=1}^{m}\frac{B_j}{\dot{V}(t_j^f)} - \frac{\partial}{\partial t_j^f}V(t^*) \tag{52}$$

Now, $\vartheta_i^{*\prime}$ can be calculated by inserting Eq. 48 into Eq. 44, which yields:

$$\vartheta_i^{*\prime} = -\frac{B_*}{A_*} \tag{53}$$

In order to calculate B_* and A_*, all times at which membrane potential reaches the threshold need to be considered: $t_x \in \{t_j^1, t_j^2 ..., t_j^m, t^*\}$. At every spike time the membrane voltage equation can be reduced to:

$$V(t_x) = \frac{V_o(t_x)}{C_{t_x}} \tag{54}$$

where

$$C_{t_x} \equiv 1 + \sum_{t_{\mathrm{j}}^f < t_x} \exp\left(-\frac{t_x - t_{\mathrm{j}}^f}{\tau_m}\right) \tag{55}$$

This gives the following derivatives:

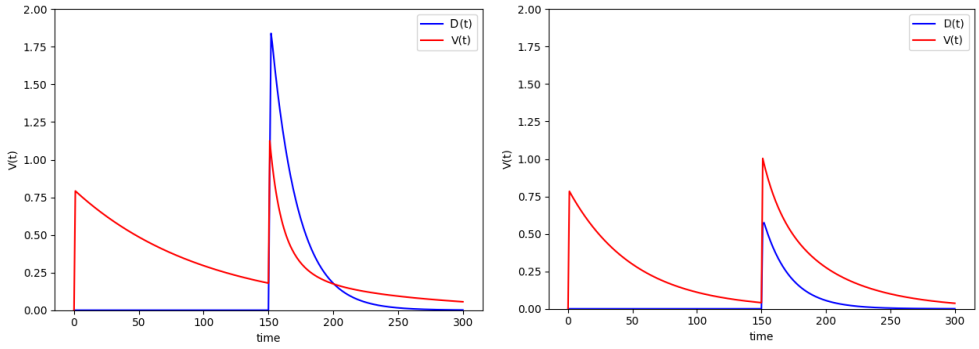
$$\frac{\partial}{\partial w_i} V(t^*) = \frac{1}{C_{t_x}} \frac{\partial}{\partial w_i} V_o(t_x) = \frac{1}{C_{t_x}} \sum_{t_i^f < t_x} K(t_x - t_i^f) \tag{56}$$

$$\frac{\partial}{\partial t_j^k} V(t_x) = \frac{-V_o(t_x)}{C_{t_x}^2} \frac{\exp(-\frac{t_x - t_j^f}{\tau_m})}{\tau_m} \text{ for } t_j^k < t_x \tag{57}$$

$$\dot{V}(t_x) = \frac{1}{C_{t_x}^2} \left(C_{t_x} \frac{\partial}{\partial t_x} V_o(t_x) + \frac{V_o(t_x)}{\tau_m} \sum_{t_j^f < t_x} \exp\left(-\frac{t_x - t_j^f}{\tau_m}\right) \right) \tag{58}$$

Appendix B

Examples of GNM dynamics.



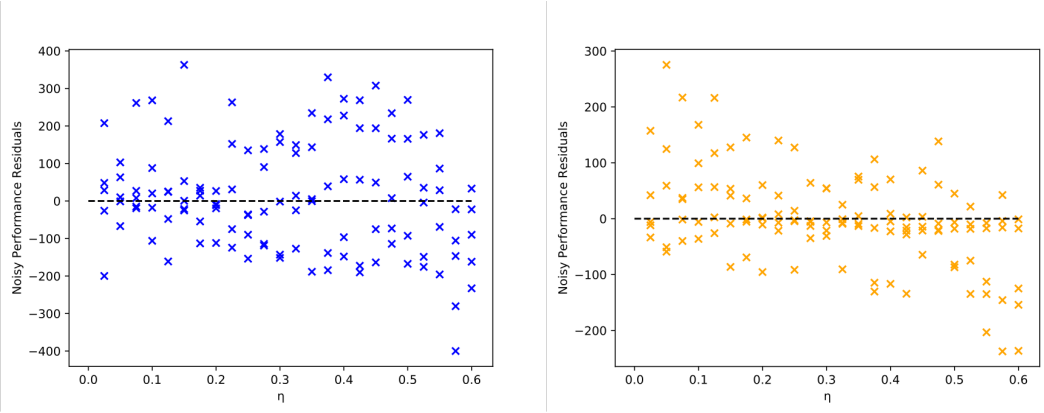
(a) $\eta = 0.8$ (b) $\eta = 0.6$

(c) $\eta = 0.4$ (d) $\eta = 0.2$

Figure 60: GNM dynamics for varied η and other parameters set in the following way: $\alpha = 0.05$, $\beta = 0.05$ and $h = 100$. Here, the GNM is presented with one sub-threshold ($t = 0$) and one super-threshold input ($t = 150$). Increasing the "spikiness" parameter η results in a more pronounced reaction to the membrane potential crossing the behavioural threshold ϑ_B.

Appendix C

GNM noisy residuals for two and four input patterns.



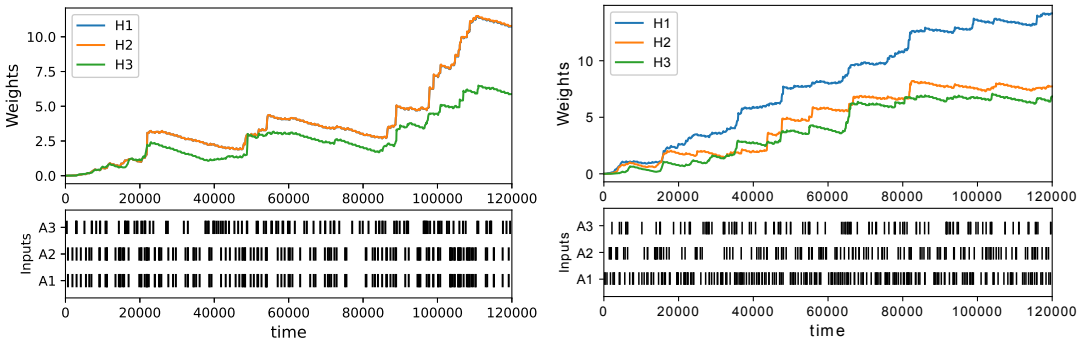
(a) Two classes of patterns

(b) Four classes of patterns

Figure 61: Noisy performance residuals for learning of two and four spatio-temporal patterns in the GNM, for the same setup as in fig. 15.

Appendix D

Detailed compilation mode in DSD



(a) Example of temporal correlation learning.

(b) Example of frequency bias learning.

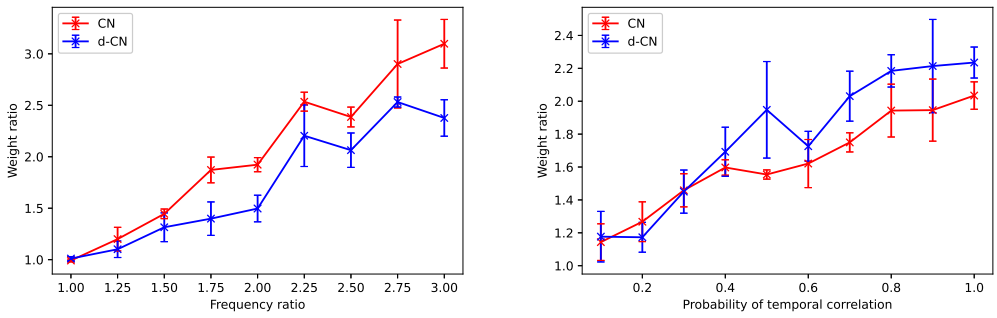
Figure 62: Example learning episode for (a) temporal correlation task, and (b) frequency bias task simulated using a detailed compilation mode of Visual DSD. The detailed mode is the most realistic setting, which assumes that binding, unbinding and branch migration have finite rates.

Appendix E

Performance of the CN and DNA neuron

When using the CN to detect a FB, the difference between the steady state weight abundances will reflect the difference of the frequencies with which the input channels fired, although the exact relationship between the two is not immediately clear. In order to understand this better, I considered a CN with three input channels. The frequency of channel 1 was varied while keeping the input frequency to channel 2 fixed. I then recorded the ratio w_1/w_2 as a function of f_1/f_2. Figure 63a shows that the weight ratio was proportional to the frequency ratio. While it remains unclear to what extent this qualitative result generalises to more complicated cases, it is apparent that CN is able to detect very small biases, albeit with a correspondingly small output signal strength.

An equivalent analysis was also conducted for the TC task. Here, I varied the probability of an input spike in channel 1 being followed by an input spike in channel 2, while keeping the total frequency of all input channels constant. So, for example, a probability of 0.5 means that on average every second input spike of channel 1 is followed by an input spike of channel 2 after a delay of δ and half of the input spikes of channel 2 occur at random times. Again, figure 63b shows that even for small probabilities, there is a reliable difference in weights between the first and the second channel.



(a) (b)

Figure 63: The responses of the CN and DNA neuron, as measured by the ratio of the first and the second channel weight, as a function of the signal strength. (a) We kept the frequency of the first channel fixed at 1 Hz and decreased the frequency of the second channel. The non-linearity was set to $m = 1$. (b) Both input channels have the same frequency of 1 Hz, but we varied the probability that an output spike of the second channel follows an input to channel 1. For both models the nonlinearity was set to $m = 5$.

Appendix F

Specification for the DNA neuron

This appendix contains the Visual DSD specification code for simulating the DNA neuron (as described in section 5.3).

```
directive rendering {classic = {mode = nucleotides}}
directive simulator deterministic
directive deterministic {stiff=true}
directive units{concentration=uM}
directive compilation infinite
directive parameters [Ain = 1; Ni = 10000.0; NiSI = 10000.0;
NiAF = 10000.0; NdegE = 12; NdegH = 0.1; Hs = 0;
 kdegE = 0.1; kdegH = 0.00002]

//// Toehold domain reactivity
dom ta = {bind=0.01; unbind=10; colour="green"} // Inputs
dom th = {bind=0.01; unbind=10; colour="orange"} // Weights
dom tb = {bind=1; unbind=10; colour="#00fbff"} // State
dom te = {bind=0.05; unbind=10; colour="#eb34e8"} // Activation
dom tfsi = {bind=1; unbind=10; colour="black"} // Fuel for SI
dom tfaf = {bind=1; unbind=10; colour="#05f796"} // Fuel for AF
//// Translator toeholds reactivity
```

```
18 dom tiwa1 = {bind=1; unbind=10; colour="#ffe000"} // WA1
19 dom tiwa2 = {bind=1; unbind=10; colour="#ffe000"} // WA2
20 dom tiwa3 = {bind=1; unbind=10; colour="#ffe000"} // WA3
21 dom tism = {bind=1; unbind=10; colour="blue"} // SM
22 dom tisi = {bind=1; unbind=10; colour="red"} // SI
23 dom tiaf = {bind=1; unbind=10; colour="#b9bdbb"} // AF
24
25 //// Inputs
26 def A1() = <ta^ a1>
27 def A2() = <ta^ a2>
28 def A3() = <ta^ a3>
29 //// Weights
30 def H1() = <th^ h1>
31 def H2() = <th^ h2>
32 def H3() = <th^ h3>
33 //// State
34 def B() = <tb^ b>
35 //// Activation
36 def E() = <te^ e>
37 //// Join gate (R1 + R2 <-> T)
38 def Join(ta, a, tb, b, tr) = {ta^*}[a tb^]:[b tr^]:[i]
39 //// Fork gate (T <-> P1 + P2)
40 def Fork(ta, a, tb, b, tr) = [i]:[ta^ a]:[tb^ b]{tr^*}
41 //// Decay modules
42 def degE() = {te^*}[e]        // E removal
43 def degH1() = {th^*}[h1]     // H1 removal
44 def degH2() = {th^*}[h2]     // H2 removal
45 def degH3() = {th^*}[h3]     // H3 removal
46
47 //// Weight accumulation: An + E <-> E + Hn
```

```
48 def WA_fuel(an, hn, tiwan) =
49 ( Ni Join(ta, an, te, e, tiwan)
50 | Ni Fork(th, hn, te, e, tiwan)
51 | Ni <hn te^>
52 | Ni <i th^>
53 | Ni <tiwan^ i>
54 )
55 //// Signal modulation: An + Hn <-> Hn + B
56 def SM_fuel(an, hn) =
57 ( Ni Join(ta, an, th, hn, tism)
58 | Ni Fork(tb, b, th, hn, tism)
59 | Ni <b th^>
60 | Ni <i tb^>
61 | Ni <tism^ i>
62 )
63 //// Signal integration: I + An <-> An + B
64 def SI_fuel(in, an) =
65 ( NiSI Join(tfsi, in, ta, an, tisi)
66 | NiSI Fork(tb, b, ta, an, tisi)
67 | NiSI <b ta^>
68 | NiSI <tfsi^ in>
69 | Ni <i tb^>
70 | Ni <tisi^ i>
71 )
72 //// Activation function: Faf + B <-> Faf + E
73 def AF() =
74 ( Ni Join(tb, b, tfaf, ib, tiaf)
75 | Ni Fork(te, e, tfaf, ib, tiaf)
76 | Ni <e tfaf^>
77 | Ni <tfaf^ ib>
```

```
78 | NiAF <i te^>
79 | NiAF <tiaf^ i>
80 )
81
82 //// Fuels N=1
83 ( SM_fuel(a1, h1)
84 | SI_fuel(i1, a1)
85 | WA_fuel(a1, h1, tiwa1)
86 //// Fuels N=2
87 | SM_fuel(a2, h2)
88 | SI_fuel(i2, a2)
89 | WA_fuel(a2, h2, tiwa2)
90 //// Fuels N=3
91 | SM_fuel(a3, h3)
92 | SI_fuel(i3, a3)
93 | WA_fuel(a3, h3, tiwa3)
94 //// Fuels Activation function
95 | AF()
96 )
```

Appendix G

Specification for the DNA neuron - B

This appendix contains the Visual DSD specification code for simulating the DNA neuron with a tuneable activation function (as described in section 5.6).

```
directive simulation {final=20000; plots=[ E0(); B(); L();
    H1(); H2(); H3(); A1(); A2(); A3() ]}

directive simulator deterministic
directive deterministic {stiff=true}
directive units{concentration=uM}
directive compilation infinite
directive parameters [bOUT=2; Ein=5; AFgateIn=10; backIn=10;
    rateb=1; tedeg=1; t1b_deg =0.1; NovelB1=10; Ain = 10; SMFin
    = 0.5; WAFin = 0.5; SIFin = 0.5; AFFin=1; NdegE = 0; NdegH
    = 0.05; Hs = 0; kdegB = 0.1; kdegE = 0.1; kdegH = 0.000005]

//// Toehold domain reactivity
dom ta = {bind=1; unbind=10; colour="green"} // Inputs
dom th = {bind=0.001; unbind=10; colour="orange"} // Weights
dom tb = {bind=1; unbind=10; colour="#00fbff"} // State

```

```
dom te0 = {bind=5; unbind=10; colour="#eb34e8"} // Activation
dom te1 = {bind=5; unbind=10; colour="#eb34e8"} // Activation
dom te2 = {bind=5; unbind=10; colour="#eb34e8"} // Activation
dom te3 = {bind=5; unbind=10; colour="#eb34e8"} // Activation
dom te4 = {bind=5; unbind=10; colour="#eb34e8"} // Activation
dom te5 = {bind=5; unbind=10; colour="#eb34e8"} // Activation

dom tfsi = {bind=1; unbind=10; colour="black"} // Fuel for SI
//// Translator toeholds reactivity
dom tiwa1 = {bind=1; unbind=10; colour="#ffe000"} // WA1
dom tiwa2 = {bind=1; unbind=10; colour="#ffe000"} // WA2
dom tiwa3 = {bind=1; unbind=10; colour="#ffe000"} // WA3

dom tism = {bind=1; unbind=10; colour="blue"} // SM
dom tisi = {bind=1; unbind=10; colour="red"} // SI

//// Inputs
def A1() = <ta^ a1>
def A2() = <ta^ a2>
def A3() = <ta^ a3>
//// Weights
def H1() = <th^ h1>
def H2() = <th^ h2>
def H3() = <th^ h3>
//// State
def B() = <tb^ b>
//// Activation
def E0() = <te0^ b>
def E1() = <te1^ b>
def E2() = <te2^ b>
```

```
44 def E3() = <te3^ b>
45 def E4() = <te4^ b>
46
47 //def L() = <b te1^ b>
48 //def L() = <b te2^ b>
49 //def L() = <b te3^ b>
50 //def L() = <b te4^ b>
51 def L() = <b te5^ b>
52
53 def nE0() = <b te0^>
54 def nE1() = <b te1^>
55 def nE2() = <b te2^>
56 def nE3() = <b te3^>
57 def nE4() = <b te4^>
58
59 def nB() = <b tb^>
60
61 //// Join gate (R1 + R2 <-> T)
62 def Join(ta, a, tb, b, tr) = {ta^*}[a tb^]:[b tr^]:[i]
63 //// Fork gate (T <-> P1 + P2)
64 def Fork(ta, a, tb, b, tr) = [i]:[ta^ a]:[tb^ b]{tr^*}
65 def Fork_WA(ta, a, tb, b, tr) = [i]:[ta^ a]:<b>[tb^ b]{tr^*}
66 //// Decay modules
67 def degB() = {tb^*}[b]                    // B removal
68 def degH1() = {th^*}[h1]                  // H1 removal
69 def degH2() = {th^*}[h2]                  // H2 removal
70 def degH3() = {th^*}[h3]                  // H3 removal
71
72 //// Weight accumulation: An + Ex <-> Ex + Hn (OLD) mx
73 def WA_fuel_mx(an, hn, tiwan, fuel, time) =
```

```
74 ( fuel Join(ta, an, te5, b, tiwan) @ time
75 | fuel Fork_WA(th, hn, te5, b, tiwan) @ time
76 | fuel <hn te5^> @ time
77 | fuel <i th^> @ time
78 | fuel <tiwan^ i> @ time
79 )
80
81 //// Signal modulation: An + Hn <-> Hn + B
82 def SM_fuel(an, hn, fuel, time) =
83 ( fuel Join(ta, an, th, hn, tism) @ time
84 | fuel Fork(tb, b, th, hn, tism) @ time
85 | fuel <b th^> @ time
86 | fuel <i tb^> @ time
87 | fuel <tism^ i> @ time
88 )
89 //// Signal integration: I + An <-> An + B - OLD
90 def SI_fuel(in, an, fuel, time) =
91 ( fuel Join(tfsi, in, ta, an, tisi) @ time
92 | fuel Fork(tb, b, ta, an, tisi) @ time
93 | fuel <b ta^> @ time
94 | fuel <tfsi^ in> @ time
95 | fuel <i tb^> @ time
96 | fuel <tisi^ i> @ time
97 )
98
99 //// Activation function: B + E -> L (m=5)
100 def AF_newE_5(fuel_gate, time) =
101 ( fuel_gate {tb^*}[b te0^]:[b tb^]:[b te1^]:[b tb^]:[b
      te2^]:[b tb^]:[b te3^]:[b tb^]:[b te4^]:[b te5^]<b> @ time
102 )
```

```
103
104 //// Activation function: B + E -> L (m=4)
105 def AF_newE_4(fuel_gate, time) =
106 ( fuel_gate {tb^*}[b te0^]:[b tb^]:[b te1^]:[b tb^]:[b
      te2^]:[b tb^]:[b te3^]:[b te4^]<b> @ time
107 )
108
109 //// Activation function: B + E -> L (m=3)
110 def AF_newE_3(fuel_gate, time) =
111 ( fuel_gate {tb^*}[b te0^]:[b tb^]:[b te1^]:[b tb^]:[b
      te2^]:[b te3^]<b> @ time
112 )
113
114 //// Activation function: B + E -> L (m=2)
115 def AF_newE_2(fuel_gate, time) =
116 ( fuel_gate {tb^*}[b te0^]:[b tb^]:[b te1^]:[b te2^]<b> @ time
117 )
118
119 //// Activation function: B + E -> L (m=1)
120 def AF_newE_1(fuel_gate, time) =
121 ( fuel_gate {tb^*}[b te0^]:[b te1^]<b> @ time
122 )
123
124 //// Short example of randomly generated input with frequency
      bias
125 def TemporalCorrelationDetection() =
126 ( 0 B() | B() ->{bOUT}
127 | Ein E0()
128 | Ein E1()
129 | Ein E2()
```

```
130 | Ein E3()
131 | Ein E4()
132
133 | 0 H1() | H1() ->{kdegH}
134 | 0 H2() | H2() ->{kdegH}
135 | 0 H3() | H3() ->{kdegH}
136
137 | backIn nE0()
138 | backIn nE1()
139 | backIn nE2()
140 | backIn nE3()
141 | backIn nE4()
142
143 | backIn nB()
144 | 0 L()
145 | AF_newE_5(AFgateIn, 0)
146
147 | SM_fuel(a1, h1, 25000, 0)
148 | SI_fuel(i1, a1, 80000, 0)
149 | WA_fuel_mx(a1, h1, tiwa1, 10000, 0)
150
151 | SM_fuel(a2, h2, 25000, 0)
152 | SI_fuel(i2, a2, 80000, 0)
153 | WA_fuel_mx(a2, h2, tiwa2, 10000, 0)
154
155 | SM_fuel(a3, h3, 25000, 0)
156 | SI_fuel(i3, a3, 80000, 0)
157 | WA_fuel_mx(a3, h3, tiwa3, 10000, 0)
158
159 | Ain A1() @ 5623
```

```
160 | Ain A2() @ 5633
161 | Ain A1() @ 8318
162 | Ain A2() @ 8328
163 | Ain A1() @ 12691
164 | Ain A2() @ 12701
165 | Ain A3() @ 13335
166 | Ain A1() @ 16757
167 | Ain A2() @ 16767
168 )
169
170 // Input signals
171 ( TemporalCorrelationDetection()
172 )
```

Bibliography

Abbott, L., DePasquale, B. and Memmesheimer, R.-M. (2016). Building functional networks of spiking model neurons. *Nature Neuroscience*, 19, pp. 350–355.

Adamatzky, A., Fullarton, C., Phillips, N., De Lacy Costello, B. and C. Draper, T. (2019). Thermal switch of oscillation frequency in belousov–zhabotinsky liquid marbles. *Royal Society Open Science*, 6(4), p. 190078.

Alon, U. (2019). *An Introduction to Systems Biology: Design Principles of Biological Circuits*. Chapman & Hall/CRC Computational Biology Series, CRC Press LLC.

Alon, U., Surette, M. G., Barkai, N. and Leibler, S. (1999). Robustness in bacterial chemotaxis. *Nature*, 397(6715), pp. 168–171.

Amos, M. (2004). *Cellular Computing*. Oxford University Press.

Antebi, Y. E. et al. (2017). Combinatorial signal perception in the bmp pathway. *Cell*, 170(6), p. 1184—1196.e24.

Ausländer, S., Wieland, M. and Fussenegger, M. (2012). Smart medication through combination of synthetic biology and cell microencapsulation. *Metabolic Engineering*, 14(3), pp. 252–260, synthetic Biology: New Methodologies and Applications for Metabolic Engineering.

Badelt, S. et al. (2020). A domain-level dna strand displacement reaction enumerator

allowing arbitrary non-pseudoknotted secondary structures. *Journal of The Royal Society Interface*, 17(167), p. 20190866.

Banda, P., Teuscher, C. and Stefanovic, D. (2014). Training an asymmetric signal perceptron through reinforcement in an artificial chemistry. *Journal of The Royal Society Interface*, 11(93), p. 20131100.

Bennett, C. H. (1982). The thermodynamics of computation—a review. *International Journal of Theoretical Physics*, 21(12), pp. 905–940.

Berleant, J. et al. (2018). Automated sequence-level analysis of kinetics and thermodynamics for domain-level dna strand-displacement systems. *Journal of The Royal Society Interface*, 15(149), p. 20180107, https://royalsocietypublishing.org/doi/pdf/10.1098/rsif.2018.0107.

Blair, H. A. (1932). On the intensity-time relations for stimulation by electric currents. *Journal of General Physiology*, 15(6), pp. 709–729, https://rupress.org/jgp/article-pdf/15/6/709/1234679/709.pdf.

Blount, D., Banda, P., Teuscher, C. and Stefanovic, D. (2017). Feedforward chemical neural network: An in silico chemical system that learns xor. *Artificial Life*, 23(3), pp. 295–317.

Bray, D. (2003). Molecular networks: the top-down view. *Science*, 301 5641, pp. 1864–5.

Brette, R. (2015). Philosophy of the Spike: Rate-Based vs. Spike-Based Theories of the Brain. *Frontiers in Systems Neuroscience*, 9(November), pp. 1–14.

Brown, T. B. et al. (2020). Language models are few-shot learners. 2005.14165.

Brunel, N. and van Rossum, M. (2007). Lapicque's 1907 paper: from frogs to integrate-and-fire. *Biological Cybernetics*, 97(5), pp. 337–339.

Cardelli, L. (2010). Two-domain DNA strand displacement. In S. B. Cooper, P. Panangaden and E. Kashefi, eds., *Proceedings Sixth Workshop on Developments in Computational Models: Causality, Computation, and Physics, DCM 2010, Edinburgh, Scotland, 9-10th July 2010*, *EPTCS*, vol. 26, pp. 47–61.

Chatterjee, G., Dalchau, N., Muscat, R. A., Phillips, A. and Seelig, G. (2017). A spatially localized architecture for fast and modular dna computing. *Nature Nanotechnology*, 12(9), p. 920.

Chen, M. and Xu, J. (2015). Construction of a genetic conditional learning system in escherichia coli. *Science China Information Sciences*, 58(11), pp. 1–6.

Chen, Y.-J. et al. (2013). Programmable chemical controllers made from dna. *Nature Nanotechnology*, 8(10), p. 755–762.

Cherry, K. M. and Qian, L. (2018). Scaling up molecular pattern recognition with dna-based winner-take-all neural networks. *Nature*, 559(7714), pp. 370–376.

Chu, D. (2017). Limited by sensing - a minimal stochastic model of the lag-phase during diauxic growth. *Journal of Theoretical Biology*, 414, pp. 137–146.

Chu, D. (2018). Performance limits and trade-offs in entropy-driven biochemical computers. *Journal of Theoretical Biology*, 443, pp. 1–9.

Chu, D. and Barnes, D. (2016). The lag-phase during diauxic growth is a trade-off between fast adaptation and high growth rate. *Scientific Reports*, 6, p. 25191.

Chu, D., Zabet, N. and Mitavskiy, B. (2009). Models of transcription factor binding: Sensitivity of activation functions to model assumptions. *Journal of Theoretical Biology*, 257(3), pp. 419 – 429.

Dalchau, N. et al. (2018). Computing with biological switches and clocks. *Natural Computing*, 17(4), pp. 761–779.

Davies, M. et al. (2018). Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro*, 38(1), pp. 82–99.

DeBole, M. V. et al. (2019). Truenorth: Accelerating from zero to 64 million neurons in 10 years. *Computer*, 52(5), pp. 20–29.

Douglas, S. M., Bachelet, I. and Church, G. M. (2012). A logic-gated nanorobot for targeted transport of molecular payloads. *Science*, 335(6070), pp. 831–834.

Fanselow, M. and Poulos, A. (2005). The neuroscience of mammalian associative learning. *Annual Review of Psychology*, 56(1), pp. 207–234, pMID: 15709934, https://doi.org/10.1146/annurev.psych.56.091103.070213.

Fedus, W., Zoph, B. and Shazeer, N. (2021). Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. 2101.03961.

Feldman, D. E. (2012). The spike-timing dependence of plasticity. *Neuron*, 75(4), pp. 556–571.

Fernando, C. et al. (2009). Molecular circuits for associative learning in single-celled organisms. *Journal of The Royal Society Interface*, 6(October 2008), pp. 463–469.

Fil, J. and Chu, D. (2020). Minimal spiking neuron for solving multilabel classification tasks. *Neural Computation*, 32(7), pp. 1408–1429.

Florian, R. V. (2012). The chronotron: A neuron that learns to fire temporally precise spike patterns. *PLoS ONE*, 7(8).

Fontana, W. (2006). Pulling strings. *Science*, 314(5805), pp. 1552–1553, https://science.sciencemag.org/content/314/5805/1552.full.pdf.

Furber, S. and Bogdan, P. (2020). *SpiNNaker: A Spiking Neural Network Architecture*.

Genot, A., Fujii, T. and Rondelez, Y. (2013). Scaling down dna circuits with competitive neural networks. *Journal of The Royal Society Interface*, 10.

Gerstner, W. and Kistler, W. M. (2002a). Mathematical formulations of hebbian learning. *Biological Cybernetics*, 87(5-6), pp. 404–415.

Gerstner, W. and Kistler, W. M. (2002b). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press.

Gillespie, D. T. (1976). A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4), pp. 403–434.

Goldt, S. and Seifert, U. (2017). Stochastic thermodynamics of learning. *Physical Review Letters*, 118(1).

Govern, C. and ten Wolde, P. (2014a). Energy dissipation and noise correlations in biochemical sensing. *Physical Review Letters*, 113(25), p. 258102.

Govern, C. and ten Wolde, P. R. (2014b). Optimal resource allocation in cellular sensing systems. *PNAS*, 111(49), pp. 17486–17491.

Grah, R. and Friedlander, T. (2020). The relation between crosstalk and gene regulation form revisited. *PLOS Computational Biology*, 16(2), pp. 1–24.

Grant, P. K. et al. (2016). Orthogonal intercellular signaling for programmed spatial behavior. *Molecular systems biology*, 12(1), p. 849.

Groves, B. et al. (2015). Computing in mammalian cells with nucleic acid strand exchange. *Nature Nanotechnology*, 11(3), pp. 287–294.

Gütig, R. (2014). To spike, or when to spike? *Current Opinion in Neurobiology*, 25, pp. 134–139.

Gütig, R. (2016). Spiking neurons can discover predictive features by aggregate-label learning. *Science*, 351(6277), pp. aab4113–aab4113.

Gütig, R. and Sompolinsky, H. (2006). The tempotron: a neuron that learns spike timing-based decisions. *Nature Neuroscience*, 9(3), pp. 420–428.

Hasler, J. and Marr, H. (2013). Finding a roadmap to achieve large neuromorphic hardware systems. *Frontiers in Neuroscience*, 7, p. 118.

Hebb, D. O. (1950). The organization of behavior: A neuropsychological theory. *Science Education*, 34(5), pp. 336–337, https://onlinelibrary.wiley.com/doi/pdf/10.1002/sce.37303405110.

Hertz, J., Krogh, A. and Palmer, R. (1991). *Introduction To The Theory Of Neural Computation*, vol. 44.

Hjelmfelt, A., Weinberger, E. D. and Ross, J. (1991). Chemical implementation of neural networks and turing machines. *Proceedings of the National Academy of Sciences*, 88(24), pp. 10983–10987, https://www.pnas.org/content/88/24/10983.full.pdf.

Hjelmfelt, A., Weinberger, E. D. and Ross, J. (1992). Chemical implementation of finite-state machines. *Proceedings of the National Academy of Sciences*, 89(1), pp. 383–387, https://www.pnas.org/content/89/1/383.full.pdf.

Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4), p. 500–544.

Hoffer, S. M., Westerhoff, H. V., Hellingwerf, K. J., Postma, P. W. and Tommassen, J. (2001). Autoamplification of a two-component regulatory system results in "learning" behavior. *Journal of bacteriology*, 183(16), p. 4914—4917.

Höppner, S. et al. (2021). The spinnaker 2 processing element architecture for hybrid digital neuromorphic computing. 2103.08392.

Indiveri, G. et al. (2011). Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5, p. 73.

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

Johansson, R. and Birznieks, I. (2004). First spikes in ensembles of human tactile afferents code complex spatial fingertip events. *Nature Neuroscience*, 7, pp. 170–177.

Jolivet, R., J., T. and Gerstner, W. (2003). The spike response model: A framework to predict neuronal spike trains. In O. Kaynak, E. Alpaydin, E. Oja and L. Xu, eds., *Artificial Neural Networks and Neural Information Processing — ICANN/ICONIP 2003*, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 846–853.

Jung, C. and Ellington, A. D. (2014). Diagnostic applications of nucleic acid circuits. *Accounts of Chemical Research*, 47(6), pp. 1825–1835.

Khan, M. et al. (2008). Spinnaker: Mapping neural networks onto a massively-parallel chip multiprocessor. In *Proceedings of the International Joint Conference on Neural Networks—Proc Int Jt Conf Neural Networks*, United States: IEEE Computer Society, pp. 2849–2856, 2008 International Joint Conference on Neural Networks, IJCNN 2008 ; Conference date: 01-07-2008.

Lakin, M., Youssef, S., Polo, F., Emmott, S. and Phillips, A. (2011). Visual dsd: a design and analysis tool for dna strand displacement systems. *Bioinformatics (Oxford, England)*, 27(22), pp. 3211–3213.

Lakin, M. R. and Stefanovic, D. (2016). Supervised learning in adaptive dna strand displacement networks. *ACS Synthetic Biology*, 5(8), pp. 885–897.

Lakin, M. R., Youssef, S., Cardelli, L. and Phillips, A. (2012). Abstractions for dna circuit design. *Journal of The Royal Society Interface*, 9(68),

pp. 470–486, https://royalsocietypublishing.org/doi/pdf/10.1098/rsif.2011.0343.

LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

Li, X. et al. (2021). Synthetic neural-like computing in microbial consortia for pattern recognition. *Nature Communications*, 12(1), p. 3139.

Linnainmaa, S. (1976). Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2), pp. 146–160.

Maass, W. (1996). Lower Bounds for the Computational Power of Networks of Spiking Neurons. *Neural Computation*, 40, pp. 1–40.

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

Macia, J. and Sole, R. (2014). How to make a synthetic multicellular computer. *PLOS ONE*, 9(2), pp. 1–13.

Macia, J., Vidiella, B. and Solé, R. V. (2017a). Synthetic associative learning in engineered multicellular consortia. *Journal of The Royal Society Interface*, 14(129), p. 20170158, https://royalsocietypublishing.org/doi/pdf/10.1098/rsif.2017.0158.

Macia, J., Vidiella, B. and Solé, R. V. (2017b). Synthetic associative learning in engineered multicellular consortia. *Journal of The Royal Society Interface*, 14(129), p. 20170158.

McGregor, S., Vasas, V., Husbands, P. and Fernando, C. (2012). Evolution of associative learning in chemical networks. *PLoS Computational Biology*, 8(11).

Memmesheimer, R. M., Rubin, R., Ölveczky, B. P. and Sompolinsky, H. (2014). Learning Precisely Timed Spikes. *Neuron*, 82(4), pp. 925–938.

Mermillod, M., Bugaiska, A. and Bonin, P. (2013). The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in Psychology*, 4.

Mohemmed, A., Schliebs, S., Matsuda, S. and Kasabov, N. (2012). Span: Spike Pattern Association Neuron for Learning Spatio-Temporal Spike Patterns. *International Journal of Neural Systems*, 22(04), p. 1250012.

Moorman, A., Samaniego, C. C., Maley, C. and Weiss, R. (2019). A dynamical biomolecular neural network. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 1797–1802.

Neftci, E. O., Mostafa, H. and Zenke, F. (2019). Surrogate Gradient Learning in Spiking Neural Networks, pp. 1–21. `arXiv:1901.09948v1`.

Nesbeth, D. N. et al. (2016). Synthetic biology routes to bio-artificial intelligence. *Essays in Biochemistry*, 60(4), pp. 381–391.

Oesinghaus, L. and Simmel, F. C. (2019). Switching the activity of cas12a using guide RNA strand displacement circuits. *Nature Communications*, 10(1).

Oja, E. (1982). Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3), pp. 267–273.

Okamoto, M. and Hayashi, K. (1983). Optimal control mode of a biochemical feedback system. *Biosystems*, 16(3), pp. 315 – 321.

Okamoto, M., Sakai, T. and Hayashi, K. (1987). Switching mechanism of a cyclic enzyme system: role as a "chemical diode". *Bio Systems*, 21(1), p. 1—11.

Okamoto, M., Sakai, T. and Hayashi, K. (1988). Biochemical switching device realizing mcculloch-pitts type equation. *Biol Cybern*, 58(5), p. 296–299.

Plana, L. A. et al. (2020). spinnlink: Fpga-based interconnect for the million-core spinnaker system. *IEEE Access*, 8, pp. 84918–84928.

Ponulak, F. (2005). ReSuMe-new supervised learning method for Spiking Neural Networks. *Inst Control Information Engineering, Poznan Univ*, 22(2), pp. 467–510.

Qian, L. and Winfree, E. (2011). Scaling up digital circuit computation with dna strand displacement cascades. *Science*, 332(6034), pp. 1196–1201.

Qian, L., Winfree, E. and Bruck, J. (2011). Neural network computation with dna strand displacement cascades. *Nature*, 475(7356), pp. 368–372.

Racovita, A. and Jaramillo, A. (2020). Reinforcement learning in synthetic gene circuits. *Biochemical Society Transactions*, 48(4), pp. 1637–1643.

Rhodes, O. et al. (2020). Real-time cortical simulation on neuromorphic hardware. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2164), p. 20190160, `https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.2019.0160`.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6, pp. 386–408.

Rudchenko, M. et al. (2013). Autonomous molecular cascades for evaluation of cell surfaces. *Nature Nanotechnology*, 8(8), pp. 580–586.

Rumelhart, D., Hinton, G. E. and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323, pp. 533–536.

Samaniego, C. C., Moorman, A., Giordano, G. and Franco, E. (2020). Signaling-based neural networks for cellular computation. *bioRxiv*.

Schliebs, S. and Kasabov, N. (2014). *Computational Modeling with Spiking Neural Networks*, Berlin, Heidelberg: Springer Berlin Heidelberg. pp. 625–646.

Schneiker, S. et al. (2006). Genome sequence of the ubiquitous hydrocarbon-degrading marine bacterium alcanivorax borkumensis. *Nature Biotechnology*, 24(8), pp. 997–1004.

Seelig, G., Soloveichik, D., Zhang, D. Y. and Winfree, E. (2006). Enzyme-free nucleic acid logic circuits. *Science*, 314(5805), pp. 1585–1588, `https://science.sciencemag.org/content/314/5805/1585.full.pdf`.

Seifert, U. (2005). Entropy production along a stochastic trajectory and an integral fluctuation theorem. *Physical Review Letters*, 95(4).

Seifert, U. (2012). Stochastic thermodynamics, fluctuation theorems and molecular machines. *Reports on Progress in Physics*, 75(12), p. 126001.

Shirakawa, T. and Sato, H. (2013). Construction of a molecular learning network. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 17(6), pp. 913–918.

Soloveichik, D., Seelig, G. and Winfree, E. (2010). Dna as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107(12), pp. 5393–5398, `https://www.pnas.org/content/107/12/5393.full.pdf`.

Tavanaei, A. and Maida, A. S. (2017). BP-STDP: approximating backpropagation using spike timing dependent plasticity. *CoRR*, abs/1711.04214, `1711.04214`.

Taylor, M. (1973). The problem of stimulus structure in the behavioural theory of perception. *South African journal of psychology = Suid-Afrikaanse tydskrif vir sielkunde*, 3, pp. 23–45.

Tomé, T. and de Oliveira, M. (2018). Stochastic thermodynamics and entropy production of chemical reaction systems. *The Journal of Chemical Physics*, 148(22), p. 224104.

Trappenberg, T. (2010). *Fundamentals of Computational Neuroscience*. OUP Oxford.

Trosset, J.-Y. and Carbonell, P. (2015). Synthetic biology for pharmaceutical drug discovery. *Drug design, development and therapy*, 9, pp. 6285–6302.

Valadez-Godínez, S., Sossa, H. and Santiago-Montero, R. (2020). On the accuracy and computational cost of spiking neuron implementation. *Neural Networks*, 122, pp. 196 – 217.

Walters, E. T., Carew, T. J. and Kandel, E. R. (1979). Classical conditioning in aplysia californica. *Proceedings of the National Academy of Sciences*, 76(12), pp. 6675–6679, `https://www.pnas.org/content/76/12/6675.full.pdf`.

Wang, Y.-H., Wei, K. Y. and Smolke, C. D. (2013). Synthetic biology: advancing the design of diverse genetic systems. *Annual review of chemical and biomolecular engineering*, 4, pp. 69–102.

Watson, J. D. and Crick, F. H. C. (1953). Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 171(4356), pp. 737–738.

Wunderlich, T. et al. (2019). Demonstrating advantages of neuromorphic computation: A pilot study. *Frontiers in Neuroscience*, 13, p. 260.

Wunderlich, T. C. and Pehle, C. (2021). Event-based backpropagation can compute exact gradients for spiking neural networks. *Scientific Reports*, 11(1), p. 12829.

Xu, X. et al. (2021). 11 tops photonic convolutional accelerator for optical neural networks. *Nature*, 589(7840), pp. 44–51.

Yi, T.-M., Huang, Y., Simon, M. I. and Doyle, J. (2000). Robust perfect adaptation in bacterial chemotaxis through integral feedback control. *Proceedings of the National Academy of Sciences*, 97(9), pp. 4649–4653, `https://www.pnas.org/content/97/9/4649.full.pdf`.

Yu, Q., Tang, H., Tan, K. C. and Li, H. (2013). Precise-Spike-Driven synaptic plasticity: Learning hetero-association of spatiotemporal spike patterns. *PLoS ONE*, 8(11), pp. 1–16.

Yubero, P. and Poyatos, J. F. (2020). The impact of global transcriptional regulation on bacterial gene order. *iScience*, 23(4), pp. 101029–101029.

Yurke, B., Turberfield, A. J., Mills, A. P., Simmel, F. C. and Neumann, J. L. (2000). A dna-fuelled molecular machine made of dna. *Nature*, 406(6796), pp. 605–608.

Zhang, D. Y. and Winfree, E. (2009). Control of dna strand displacement kinetics using toehold exchange. *Journal of the American Chemical Society*, 131(47), pp. 17303–17314.

Zhang, L. I., Tao, H. W., Holt, C. E., Harris, W. A. and Poo, M.-m. (1998). A critical window for cooperation and competition among developing retinotectal synapses. *Nature*, 395(6697), pp. 37–44.