UTRECHT UNIVERSITY

MASTER THESIS

A cheap spiking neural model for evolved controllers

Main Supervisor:
prof dr. Serge DUMOULIN

Author:
Asgeir BERLAND

Daily Supervisor:
Meindert KAMPHUIS

Examiner:
prof. dr. A.P.J.M. SIEBES

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

in the

Faculty of Science

May 30, 2016

"If it weren't for the last minute, nothing would get done."

Rita Mae Brown

UTRECHT UNIVERSITY

Abstract

Faculty of Science

Master of Artificial Intelligence

A cheap spiking neural model for evolved controllers

by Asgeir BERLAND

When implementing spiking neural networks into agents acting in virtual environments, there often exist an underlying problem of incongruity between the level of detail in the (biological) neural model and the agent's environment, resulting in excessive network iterations required for functional behavior. This thesis aims to explore a new spiking neural model specifically made to address this issue, and will be assessed by its similarity to biological neurons, its cost in computation, and finally its behavioral capabilities when acting as a controller for a virtual creature. The results show that our new spiking model is significantly cheaper to implement, in addition to outperform other neural models in certain behavioral tests.

Acknowledgements

The author would like to thank the employees of MeinMein where he learned most of his programming skills, in addition to the vast amount of feedback received during development.

In addition, he would especially like to thank his parents for their support during the prolonged duration of this thesis.

Contents

List of Figures

List of Tables

List of Abbreviations

AP	**A**ction **P**otential
GP	**G**raded **P**otential
NCF	**N**euro-**c**omputational **F**eature
SNN	**S**piking **N**eural **N**etwork
FF	**F**eed-**F**orward
HH	**H**odgkin-**H**uxley model
SM	Izhikevich's **S**imple **M**odel
CM	**C**ontroller **M**odel
MLP	**M**ulti**l**ayer **P**erceptrons (model)
LIF	**L**eaky **I**ntegrate-and-**F**ire model
TED	**T**emporal **E**dge **D**etection
SUS	**S**tochastic **U**niversal **S**ampling

Chapter 1

Introduction

Over the last two decades in computer science, the development of artificial neural networks has been on the rise in several domains. While Google researchers have provided us with ways to extract extensive classes and styles from images[1] and outplay humans in classic games [2, 3], others have evolved controllers for virtual creatures to interact in environments [4, 5]. In addition, several games [6, 7, 8, 9] have incorporated neural networks to create interesting behavior, either to interact or compete with the player. In neuroscience, they are often used to simulate large brain areas [10], or smaller networks of biological organisms [11, 12] to get a better understanding of neural dynamics. Artificial neural networks are very powerful in their ability to be *trained* to solve complex problems that can be hard or troublesome to define specific methods for, especially when little is known about what the best solution is supposed to be.

In nature, we can witness a vast diversity of intelligent solutions to the problem of survival. This largely depends on the animal's morphology and how it interacts with acquiring resources from the environment. What all animals have in common is a form of control system to achieve this, varying from a small to large neural networks, the latter commonly being referred to as a brain when clustered. However, it is the smallest unit of information processing within these networks, namely the *neuron*, that will be the main focus of this thesis. Most of the artificial neurons that were used in the papers mentioned in the introduction are heavy abstractions of biological neurons, meaning that they tend to lack rudimentary mechanics used in neuronal computation. Still, most of the time they have proven their worth, being cheap to implement and trained with ease. However, this is not always the case for abstractions of biological neurons, as they are often heavy on computation due to more complex mechanics. Due to these different mechanics, common training methods generally needs to be revised.

In this thesis, we propose a new neural model which is intended to fill the gap between heavy and light abstraction of biological neurons, in addition to making it easy and cheap to implement. Specifically, the model's main purpose is to act as a controller for an embodied agent (a creature) within a virtual environment.

1.1 Requirements

In order to validate the quality of our neural model, the three items listed below will be used for assessment when compared with other contemporary models.

1. Richness
2. Cheapness
3. Functionality

The richness (1) refers to the amount of neuro-computational features (to be explained in chapter 2) a neuron can output given various inputs. The cheapness (2) is assessed by

how expensive the model is to simulate computationally. Finally, the functionality (3) is the quality of solutions the model can produce for behavioral problems.

In order to assess these requirements, the reader will first be introduced to biological and artificial neurons in chapter 2 which covers background knowledge required for the next chapters. In chapter 3 the new neural model is introduced, in addition to assessing its richness and cheapness requirements. In chapter 4 we describe the testbed environment, involving a virtual creature searching for food. Chapter 5 describes two behavioral experiments performed in the testbed. Each test will have a local discussion, and will together be used for assessing the functionality requirements. A final evaluation of the requirements will take place in chapter 6, along with discussion potential future research together with the thesis conclusion.

Chapter 2

Neurons and models

2.1 The biological neuron

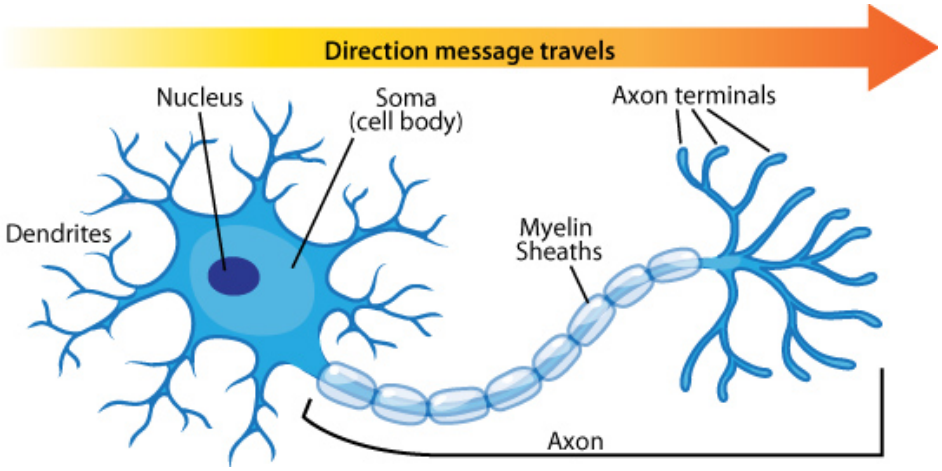


FIGURE 2.1: Nerve Cell: Dendrites receive signals from other neurons in which the cell body reacts to. Given sufficient stimulation, the cell produces a new signal which travels through the axon to finally propagate towards other receiving neurons [13]

The nerve cell, or *neuron* (Figure 2.1), is a specialized unit within a biological system which is responsible for processing information from the environment into muscle control. Neurons generally come in numerous quantities, forming organized networks which is connected through *synapses*. To get a perspective, the neural net within the nematode *Caenorhabditis elegans* (a roundworm of 1 mm in length) has been precisely mapped to 302 neurons with roughly 7500 synapses. This network is sufficient to provide it with abilities such as chemotaxi, learning, memory, and mating behavior [11, 12]. On the other side of the spectrum, the human brain has an estimate of around 86 billion neurons, complemented with the staggering amount of 10^{14} synapses connecting them. The way neurons communicate via the synapses is usually through chemical means by releasing neurotransmitters, which will be explained in the following sections. Although there also exist electrical gap junctions between neurons, they will not be focused in this thesis.

2.1.1 The Action Potential

Perhaps the most typical form of communication among neurons in animals is through the *action potential* (AP), its product often referred to as a *spike*. The event is described by a sudden peak in the neuron's membrane voltage, causing it to rise and fall over a course

of 2ms. This causes a pulse to propagate through the axon, branching throughout terminals towards dendrites of other neurons. In order for a neuron to undergo an AP, it has to be appropriately stimulated by neurotransmitters released at these terminals. Neurotransmitters are the primary stimulus for direct information processing, as well as other biochemical changes responsible for longer-term processes such as learning. Depending on the type of neurotransmitter, this will either cause a depolarization for an excitatory types, or a hyperpolarization for inhibitory types. This tends to make the neuron more or less likely to undergo an AP, depending on; the type of neuron, the time since the last AP, and the amount of neurotransmitters released. This is contrary to the myth[14] that neurons undergo APs after the membrane reaches a certain fixed threshold set in millivolts.

When a neuron capable of undergoing APs has its membrane directly stimulated with a DC current, it has a tendency to produce a so called *spike train* - a distinct succession of spikes. A spike train can reach a wide variety of patterns and frequencies, its upper bound usually reaching 200hz[1]. In order to illustrate the various spike trains (patterns) biological neurons can produce, a comprehensive map is shown in figure 2.2. External stimuli is not always required to invoke APs. Pacemaker neurons are well documented within the cortex[16, 17], able to undergo APs independent of stimulation from other neurons.

2.1.2 Graded and other potentials

There also exists neurons that do not undergo APs, instead retaining a graded potential (GP) within the membrane. These neurons do not release any distinguishable spikes, instead their membrane has a smooth curve of de/hyper-polarization rising and falling over time. This causes the release of neurotransmitters to neighboring neurons proportional to the current membrane potential. GP neurons are present in virtually all animals, often associated with sensor neurons transducing an external stimulus, or motor neurons receiving spikes from inter-neurons to tense muscles.

To give an example of such a process regarding GP, we will explain photoreceptors which responds to light from our environment. It all starts with rods and cones which are sensitive to light and color. In response to light, they start releasing the neurotransmitter glutamate (excitatory) towards neighboring bipolar cells. The amount of neurotransmitters released are directly proportional to the intensity of the stimulus (light). As a result, the bipolar cells' membrane become depolarized, and in turn further releases glutamate towards the final gatekeeper between the brain and eye; the ganglion cell[2]. A ganglion cell is considered to undergo receptor potential (or *generator* potential) because of its ability to convert the GP received from the bipolar cell(s) into APs (spikes) that propagate throughout the brain. All the information your brain interprets from visual perception is exclusively rooted in spike trains originating from these ganglion cells.

Although out the scope of this thesis, it is worth mentioning the existence of some other forms of graded potential. Plateau potentials for instance, can retain their depolarized state long after the offset of an excitatory synaptic input. This causes the neuron to remain active until a follow-up inhibitory input turns it inactive. These types of potential are especially important for spinal motor systems[20], also present in some of *C.elegans* neurons, as none of them are able to produce APs[21].

[1] For a more detailed overview of spiking frequencies, see [15]
[2] For further reading about the retina, see [19]

2.1.3 Why different potentials?

A weakness of the GP is that it the signal diminishes along with the distance traveled. This is not the case for the AP, as the spike gets innervated by the axon as it travels and propagates to neurons further away. If the ganglion cell did not produce spikes, the information from our photoreceptors would never reach the rest of our brain. There are both upsides and downsides to this conversion from GPs to APs from a theoretical standpoint. The downside is that APs are more expensive to maintain in terms energy usage compared to GPs. In addition, neurons that spike virtually ignore the input received during the APs (≈2ms), resulting in a discrete information loss[22]. This could explain why some nematodes - like the *c.elegans* - do not undergo APs. As information does not have to travel very far within its small neural network, the exclusive use of GP could make it very energy efficient. Besides the benefit of speed, another upside of the AP is that it seems to bottleneck excessive information[23], which might be something that's essential to higher cognitive states. This could be attributed to spike coding schemes in networks (to be discussed in section 2.3), providing an upper hand over GPs when it comes to complex signal integration.

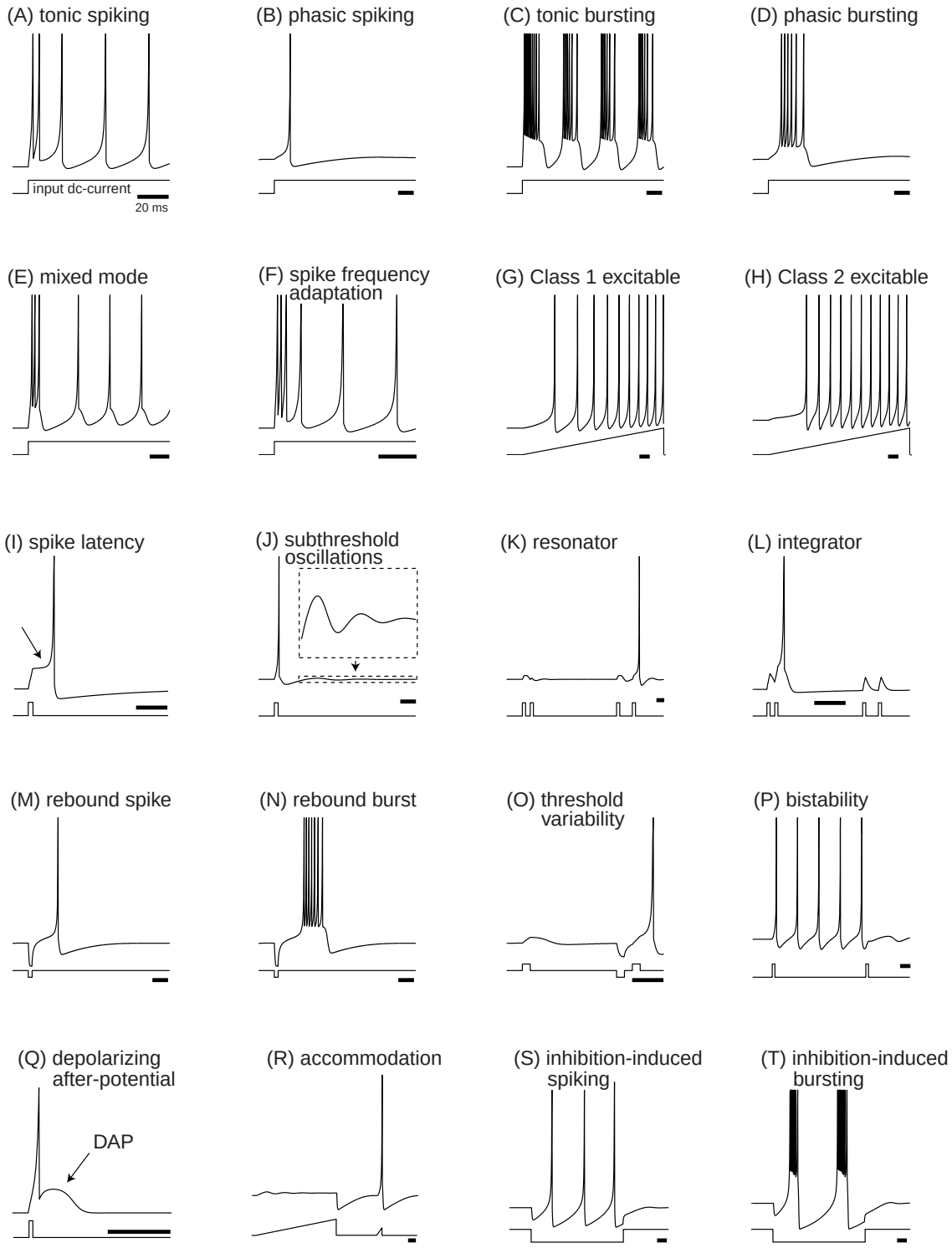


FIGURE 2.2: 20 Neuro-computational features (NCFs) of biological neurons produced by Izhikevich [18]. Spikes are shown as sharp peaks in the membrane potential, resulting from the input current shown below.

2.2 Artificial networks and neurons

Artificial neurons here are defined as a processing unit adhering to a computational model, able to take a set of inputs to produce a single output at a discrete point in time. It commonly resides in numerous amounts within a network architecture which defines how data is propagated among them. Hence, a neural network can be considered a processor which is constituted by several micro-processors. Since networks implement artificial neurons, it is easier to keep the two aspects isolated in terms of elaboration, considering various types of network architectures and neuron models are often compatible with each other.

The most common network is of a fully connected layered feed-forward (FF) architecture, shown in figure 2.3. FF is commonly defined by the data flowing throughout the network in one direction, while fully connected means there exists a connection between all the neurons in layer n and layer $n+1$. *Recurrent* networks on the other hand are more liberal with the directions of connections. For example, the last given output could be fed to the network as input in the following computational step. Interestingly, the neural net of *c.elegans* is mostly FF[24]. This gives us a good indication what is possible to achieve with such a simplistic architecture. Although out of this thesis' scope, a recent method worth mentioning is variation of FF architecture called a *convolutional* network. The gist of this method involves using a small set of neurons to process a larger set in an iterative fashion, usually in the form of a mask for a 2D visual field. Convolutional networks received a lot of attention in the recent years, often referred to as 'deep' neural networks [3, 1].

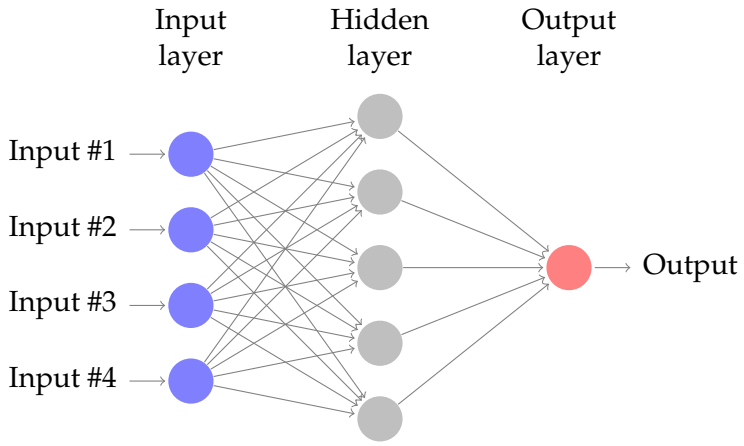


FIGURE 2.3: A fully connected feed-forward architecture of an artificial neural network.

We will now explore the various types of computational models that are commonly described as artificial neurons, in addition to their applications and limitations.

2.2.1 The Perceptron (MLP)

The *perceptron* proposed by Rosenblatt in 1928 [25] has had a major influence in the field of machine learning throughout the recent decades. Since that time it has undergone substantial refinement, and we will here present the most common model in figure 2.4.

The weight coefficient w modulates the strength of a connection, and can be either positive or negative. Analogous to the biological neuron, the weight can represent the synapse strength, with positive values representing excitatory and negative inhibitory

neurotransmitters. The bias value b (reflexive weight) ensures that the neuron is capable of producing output regardless in the absence of any input, given $b > 0$.

The activation function's role is to normalize or bottleneck the output, avoiding skewed numerals. Here we have multiple choices. For example, if we are only interested in an on-off output, we can merely define the activation function to output 1 if it is above a certain threshold value, 0 otherwise. However, most of the time we are interested in a more graded non-binary output value. The most common activation functions are sigmoid or hyperbolic tangent, the latter shown in figure 2.4. The main difference between these two is that the sigmoid output range is [0:1], where $y = 0.5$ for $x = 0$, while the hyperbolic tangent is [-1:1], where $y = 0$ for $x = 0$. The usefulness of this activation function is perhaps more apparent when we introduce several layers of perceptrons to keep the output normalized, the end result commonly referred to as multilayer perceptrons (MLP).

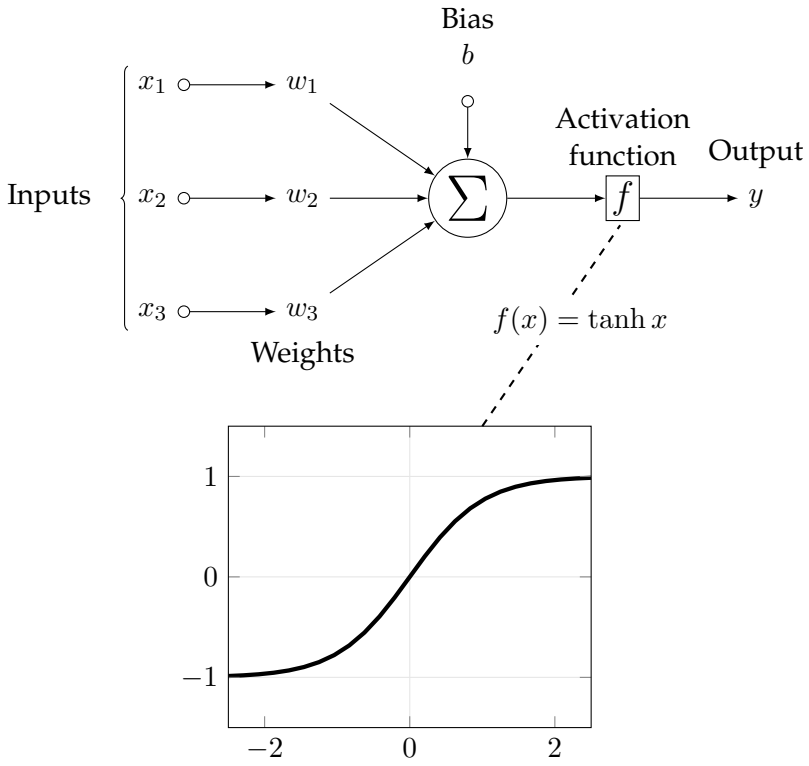


FIGURE 2.4: A perceptron. Raw inputs x_i undergo a weighting coefficient w_i before being summed along with a bias value b. Finally, the sum undergoes an activation function f which acts as a bottleneck for the final output y. In this example, the f is a hyperbolic tangent function which transforms the sum into a value within the range [-1:1].

The great strength of multiple layers opposed to a singular one makes the MLP able to solve problems that are not linearly separable. Its simplistic processing also makes the network fairly easy to train as there are few parameters (the weights and bias) in need of adjustment, often backed up with the powerful back-propagation learning algorithm [26]. However, this algorithm has limitations which will be further explained in chapter 4.2, as its best understood within the context of an environment. The perceptron's simplicity is also double-edged sword. Given that it does not have a state (memoryless), it will always produce the same output for the same input. Although there are workarounds for these issues, they are often tailored to fit a specific problem.

Still, the MLP is probably the most common form of artificial neural network to date, and it is still the preferred model for contemporary implementations, such as the convolutional networks described in the start of this section.

2.2.2 Spiking models

Contrary to the perceptron, spiking neural models put more emphasis on the underlying mechanics of biological neurons and spike production. The amount of spiking models are vast, and only a few relevant ones will be presented here in light of this thesis.

The most noteworthy contribution in this domain is undoubtedly the work of Hodgkin and Huxley [27] with their conductance based model which accurately describes how neurons undergo APs. For their extensive work, they both received the Nobel Prize in 1963. However, the HH is rather impractical for engineering purposes, given that large amounts of neurons are very expensive to simulate on contemporary hardware. Their model is primarily used for scientific purposes, and serves as a very good control for future models which attempt to replicate neuronal behavior in a cheaper fashion.

Izhikevich presents a very good review in his paper[18] comparing numerous spiking models based on their NCF and computation price, and we will here try to mention some of them. Starting with the simplest model, the leaky Integrate-and-Fire (LIF) neuron[28] operates with a capacitor and resistor on input voltage similar to the HH model. A simplified version of the LIF model[29] is shown below:

$$\tau_m \frac{du}{dt} = -u(t) + RI(t) \qquad \text{if } u > u_{thresh}, u \leftarrow u_{reset}$$

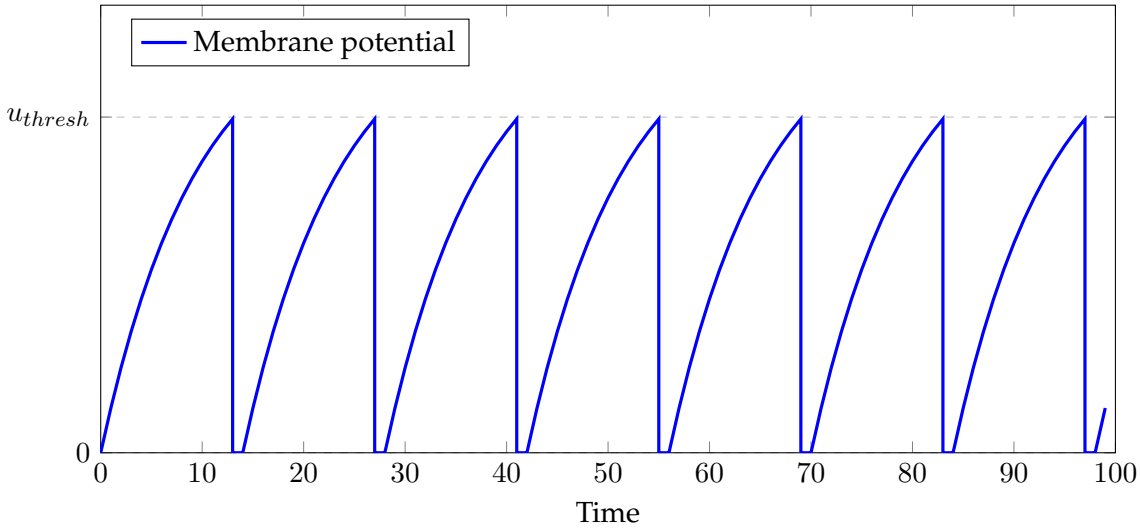


FIGURE 2.5: An example of a LIF neuron under constant input. Spikes occur when the membrane voltage supersedes the threshold value.$M_{tau} = 10$.

Here, the u represents the neuron's membrane voltage, τ_m the membrane time constant, R the linear resistor, and I the input voltage. As spikes are not explicitly generated by the model, a supplementary rule is to have it occur when the membrane reaches a specific threshold, and then set back to a fixed value (usually 0) as shown in figure 2.5. The LIF model is an important mention because it replicates the most basic feature of the biological neuron, integrating time dependent input at varying intensity to release discrete spike trains. Despite its abstraction, the model can sometimes outperform more complex models[30](including HH) at replicating spikes of biological neurons. Although

being very cheap to implement, it comes with grave limitations in regards to the few neuro-computational features (NCF) it is able to produce (3 out of 20).

Izhikevich's own model, referred to as the 'simple model' (SM)[18], is arguably one of the best with respect to cheap and realistic neuronal behavior. The SM is described by the two ordinary differential equations below, supplemented with a conditional for spike events.

$$v' = 0.04v^2 + 5v + 140 - u + I$$
$$u' = a(bv - u)$$
$$\text{if } v \geq +30mV, \text{then } \begin{cases} u \leftarrow c \\ v \leftarrow u + d \end{cases}$$

Here, v' and u' are derivatives of the membrane potential and recovery variable with respect to time. The values $5v$ and 140 are specifically chosen so the values represent time and voltage levels of biological neurons. It is important to point out that the value $+30mV$ here is not a threshold, but merely a check to detect a sudden spike of membrane potential which has consequences. By altering parameters, both the SM and HH are able to replicate all of the NCFs presented in figure 2.2. In comparison however, the SM is vastly superior due to its extremely cheap implementation (13 FLOPS vs 1200 FLOPS for 1ms simulation[18]).

2.3 Neural information processing

A widely debated topic in computational neuroscience is whether rate or spike-coding is the core of information processing within biological neural networks. Given that most neurons in animals elicit spikes as a form of communication, and that the firing rate increases along with stimuli intensity, one could make an assumption that the only important information is the rate of which spikes arrive. For example, one could scale a 0-200hz frequency to [0:1], making it more or less equivalent to our non-spiking perceptron with an appropriate activation function. However, Maass [31] has shown that spiking neural networks (SNN) have computational power far beyond these traditional MLP networks, requiring less amount of neurons for the same problem. Also, Thorpe [32] has made good arguments for how the discrete timings of individuals spikes arriving in the human visual cortex explains fast facial recognition. In his philosophical paper on this topic, Brette [33] illuminates the fact that since rates are rooted in spikes, it is ultimately the spike that holds any form of causality in information processing. Although outside the scope of this thesis, a better understanding of neural *plasticity* - how neurons form connections with each other to learn and form memories - would undoubtedly shred more light on this topic.

2.4 Neural implementation for agents

In order to prepare the reader for upcoming chapters implementing neural networks, we will here address and elaborate upon the challenges it imposes.

In physics simulations, objects tend to get moved and pushed around as a result of forces. If an object gets pushed with a certain force, we need to calculate its position throughout time to both track and visualize movement. Hence, we need to decide how often we want to calculate the object's new position, referred to as the physics's *timestep*. A common timestep for physics simulation is ≈17ms (60hz), meaning that during the course of a second the world is updated 60 times.

Agents can be part of these physics worlds, taking input from the environment to produce behavioral output. Neural networks can be implemented in these agents to act as a controller. However, models possessing states (such as membrane potential) are not always trivial to integrate with the world's physics. Specifically, we are referring to models which can produce a different output for the same input.

When working with models such the MLP, which do not have states, it is fairly simple to update the neural network in lockstep with the physics step. However, when working with spiking neural networks (SNN) on the other hand, we have to be cautions. The SM is specifically designed to model real neurons on a 1ms timescale, and is thus highly out of synchrony with the fast moving world of a 17ms timestep. To form an analogy, implementing the SM in lockstep with such a world would be as if the reader experienced their environment in fast forward (17x). Your brain would not be able to keep up with the rapid input from the environment, and your muscle neuron outputs would be lagging behind.

To fix these skewed timesteps, we have two choices. Either increase the physics timestep to 1ms in order to match the SM, or perform several network steps per physics step while tracking the output neurons' activation during the interval. The former method is extremely expensive in terms of computation if the world step is heavier than the network step, and is possibly only an option if one would desire hyper-realism of a real biological organism. The latter method is more commonly used, with [8] measuring the output neurons firing rate over 20 timesteps, and [9] over 100 timesteps. Hence, we define a network *cycle* to be the process of updating all its neurons according to the rules set by the model, while a *network update* means performing a specific amount of cycles. The optimal amount of cycles for both computational efficiency and behavioral performance depends largely on both the model and the environment.

Chapter 3

The Controller Model

3.1 Neuronal capacities

Neurons undergo complex information processing. This implies that neurons are capable of modulating the output as a result of the input, and that this modulation depends on the type of neuron. Although different neuron types can produce several NCFs depending on input, one neuron is not expected to be capable of all. This is because some features are mutually exclusive [18]. However, the tendency of a neuron to switch firing modes (from one NCF to another) when operating in a biological system is less clear. It has been documented that some can switch from spiking to bursting [34, 35]. Therefore, we expect a proper neuronal model to produce neurons that can - to some degree - switch modes as a result of differentiated input. We also expect a positive relation between the richness (NCFs) of a model, and its embodied behavioral capacity.

Axiom 1 *If a NCF is useful for a certain behavioral task, we would expect an agent implementing a model possessing this NCF to outperform others which do not.*

3.2 Development

The philosophy behind the spiking neural model about to be presented in this thesis is taken from a functionalist's perspective. This means that we will evade mechanics meant to represent biological realism. Thus, all the variables are left dimensionless as they do not represent voltage or chemical compositions.

The focus is set on the input-output relations of neurons seen in a temporal dimension. Inspired by biological neurons, we also want to blend the functionality of both graded and action potentials while retaining a rich diversity of NCFs.

Although most spiking models use integration over elective time steps with ordinary differential equations, the first iteration of the new model currently does not. Instead, it is a discrete system with countable steps executed in a sequential manner. It is specifically designed to be implemented in agents of virtual environments with large timesteps as a behavioral controller. As such, for the lack of a better name, it is dubbed the Controller Model (CM) throughout the rest of this thesis.

During development, the author's initial starting point for the CM consisted of variant LIF type neuron. From there, a dynamic threshold was added capable of adapting to the membrane potential. For final tweaking of the CM's mechanics, Izhikevich's comprehensive collection of NCFs[18] was used for validation. As a result of tweaking, the author stumbled upon an emergent behavior resulting exclusively from negative input, causing oscillating spike trains and delays. Coincidently, this behavior was very similar to some of the more complex NCFs.

3.3 Formulating the CM

We let the state of a CM neuron be $N_s = \{m, t\}$ where m is the membrane potential and t is the current threshold value. The neuron possesses 3 parameters $N_p = \{a, b, c\}$, which are all set in the restricted domain [0:1]. Here a and b are the membrane and threshold decay coefficients, while c represents the equilibrium point of t. For initialization, $m \leftarrow 0$ and $t \leftarrow c$.

$$m' \leftarrow m + I \tag{3.1}$$

$$\text{if } m' \geq t \begin{cases} t' \leftarrow t + bm' \\ m'' \leftarrow 0 \end{cases} \tag{3.2a}$$

$$\text{if } m' < t \begin{cases} m'' \leftarrow am' \\ t' \leftarrow t + bm'' \end{cases} \tag{3.2b}$$

$$t'' \leftarrow (c - t) \cdot \frac{b}{2} \tag{3.3}$$

Input I is added directly to the membrane (3.1), followed by checking whether the new membrane value exceeds the threshold (3.2a) or not (3.2b).

In the case of 3.2a, a spike occurs along with readjustment to the N_s. First, the threshold adapts to the current membrane value, given that the threshold decay coefficient $b > 0$[1]. Specifically, threshold value adapts by increasing for positive, and decreasing for negative membrane values. Secondly, the membrane snaps to its initial value of zero.

In the other case of 3.2b which produces no spikes, the membrane *first* decays towards zero given $a > 0$, followed by the threshold adapting to it.

Finally, the last step 3.3 causes the threshold to move towards its equilibrium c as a result of the distance between them. The significance of this move is a fraction of the threshold decay parameter b, as to ensure the membrane has a higher impact than the equilibrium.

It is important to notice the order of operations in 3.2a and 3.2b are flipped around respectfully in terms of implementation. This is because we want the threshold to increase right after a spike has occurred (refractory period) based on the last state of the membrane, ultimately making it less likely to fire again in the following step(s). In the latter case (3.2b), we want to move the threshold first prior to decaying the membrane. If not, preliminary experimentation of the model indicated that the threshold would often elude the membrane completely.

When implementing the model in a neural network using connections in forms of weights, there are certain precautions to be aware of. It is convenient to keep the equilibrium parameter fixed at a certain value[2] for all but the input neurons. This is because the weights of incoming connections are traditionally subjected to mutate, and thus will adapt themselves relative to the equilibrium. This greatly helps learning algorithms as they have one less parameter to tune. For input neurons however, we want the equilibrium to mutate because the inputs are graded, not spiking. For example, highly sensitive input neurons often evolve $c = 0$, meaning that they will be able to detect any form of micro-perturbations at resting state[3]. This greatly facilitates the input neurons ability to

[1] If $b = 0$, the CM neuron is similar to a LIF neuron given $a > 0$ as it can produce the same NCFs.
[2] By default $c = 0.5$ for weights ranging [-1:1].
[3] For a good example, see figure 5.6.

undergo generator potentials; they are the first neurons to transduce arbitrary data into action potentials.

To explain the dynamic system in a simplified analogous manner, we can view the membrane as a mechanical spring. The input stretches membrane over time, and will in turn be more likely to collapse back to its normal state (releasing a spike) if the stretching exceeds its threshold. The threshold in turn is a two-way spring trying to hold a relative distance with the membrane and its natural equilibrium.

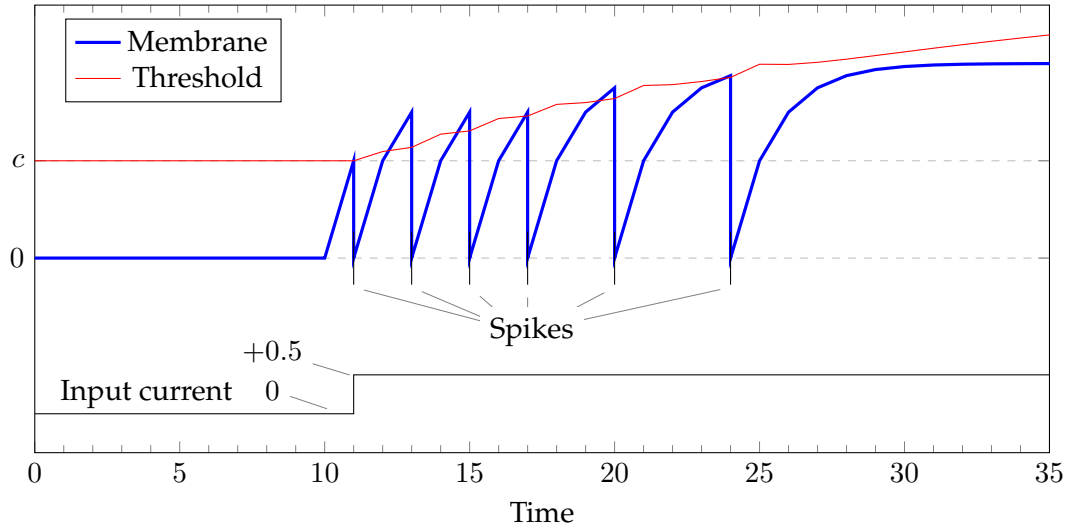


FIGURE 3.1: An example of a CM neuron performing a phasic burst output (spikes) after the onset of a constant input current. Spikes occur when the membrane value supersedes the threshold value. N_{para} = {$a = 0.5$, $b = 0.1$, $c = 0.5$}.

3.4 NCF assessment

Izhikevich himself has not made any explicit methodology for assessing whether or not an NCF has been successfully replicated by a model. Hence, judgment of the CM's capabilities will largely consist of an open discussion supplemented with illustrations. Any successful replication also requires the output to be modulated by the input[4]. This means that we expect the neuron to behave differently in a predictable manner by adjusting the input gradually.

A comprehensive set of graphs are presented in appendix A displaying 13 approximate replications of NCFs comparable with figure 2.2. An extracted example is also shown here in figure 3.1. The neural behavior shown in these graphs were discovered with relative ease through gradual tweaking of the a and b parameters. 10 of the replications behave fairly similar to biological neurons, and will not be further discussed. Instead, we will address 3 controversial, 3 possibly redundant, and 4 missing replications.

3.4.1 Controversial : N, T, P

Unlike biological neurons, the rebound burst (N) and the inhibition-induced bursting (T) replications do not have any intervals between the produced spikes. This is also the case

[4]Class 2 excitability (H) is an exception to this.

for bistability (P), which requires a negative input bias in order to function. The lack of spike interval makes it more appropriate to refer the output as phasic currents.

3.4.2 Possibly redundant : C, S, M

Despite their absence, feature C, S and M might be complemented by others. This is because the share a similar output behavior with some NCFs when receiving an inverted input. This inversion is easily achievable by flipping the sign of incoming weight to the neuron. Tonic spiking (A) is the inverse of rebound spike (M) and inhibition-induced spiking (S). Inhibition-induced bursting (T) is the inverse of phasic bursting (C). Hence the features are possibly redundant functionally, unless we require the neuron to switch between *specific* firing modes (NCFs) as a result of input differentiation.

3.4.3 Missing : I, J, K, Q

Although spike latency (I) is not present for individual spikes, delayed output is still possible in the form of rebound bursts (N). It is worth mentioning that generic spike delays tend to be modulated by weak sub-threshold input [18], and these small delays might not be relevant for large timesteps.

Subthreshold oscillations (J), and resonator (K) are highly related. This is because the resonator requires membrane voltage oscillations, making it sensitive to certain frequencies of incoming spike trains [36]. This makes it act as a filter for certain bands of frequencies. However, these bands are greatly limited when we operate in discrete timesteps. Even if resonating neurons were possible with the CM, they would be capped by the large timesteps in virtual environments. The role of resonating neurons in biological systems are poorly understood, although a recent paper [37] has shown that it might relate to differentiate feelings experienced by the cerebral cortex.

Depolarizing after-potential (Q) makes the neuron more likely to fire if it has recently fired a spike. Although the CM model does not support this, it might be achievable by inverting the threshold movement relative to the membrane value when spiking.

3.5 Performance assessment

In order to get an idea of how cheap the CM is to implement, we will benchmark it to compare against MLP and the SM. First however, we will elaborate upon the specific implementations per model as it affects benchmark results. The amount of cycles (explained in section 2.4) required for a network update is based on a virtual physics environment operating at 60hz, further described in the next chapter.

3.5.1 Model implementation

The MLP implementation is identical to that of section 2.2.1, using a hyperbolic tangent function as it is faster to process than the sigmoid function [38]. Considering the MLP being stateless, it does not require more than 1 cycle for a network update.

The CM implementation is identical to that of section 3.3, with the amount of cycles set to 3 for a network update. This set amount was a result of preliminary testing in an environment to be discussed in the next chapter.

Lastly, the SM implementation differs slightly from the one explained in the end of section 2.2.2. Numerical errors can occur within the SM when the timestep is $\tau = 1ms$ and the input $I < -45$, causing the voltage v to reach infinity. Izhikevich himself seem to handle this by using a modified Euler method (midpoint) in his implementation [39],

although this is more computationally expensive. Since numerical errors are undesired, the same implementation used by Izhikevich will be used here. The amount of cycles is set to 20, guided by both preliminary testing and literature [8].

3.5.2 Setup

Benchmark *A* measures the timings of network updates for the three models. Specifically, we simulate 1000 randomized networks (multi-threaded) with 1000 hidden neurons each in a FF architecture. To ensure data flow, the MLP network receives a constant input value to propagate towards the hidden neurons, while for the SM and CM a single input neuron receives the same input value (scaled for the SM). The output layer consists of only 3 neurons.

Benchmark *B* measure the timings of neuron updates under conditional input, independent of connections and spike consequences occurring in networks. The perceptron (MLP) is excluded from this test as it does not update a neural state. 1000 neurons with randomized parameters are updated 1000 times, averaged over 1000 trials. Both the models CM and SM were under the influence of excitatory (0.5, 20), inhibitory (-0.5, -20), and silent input currents.

All models were implemented with a data-oriented design on the same hardware[5], coded in D.

3.5.3 Performance

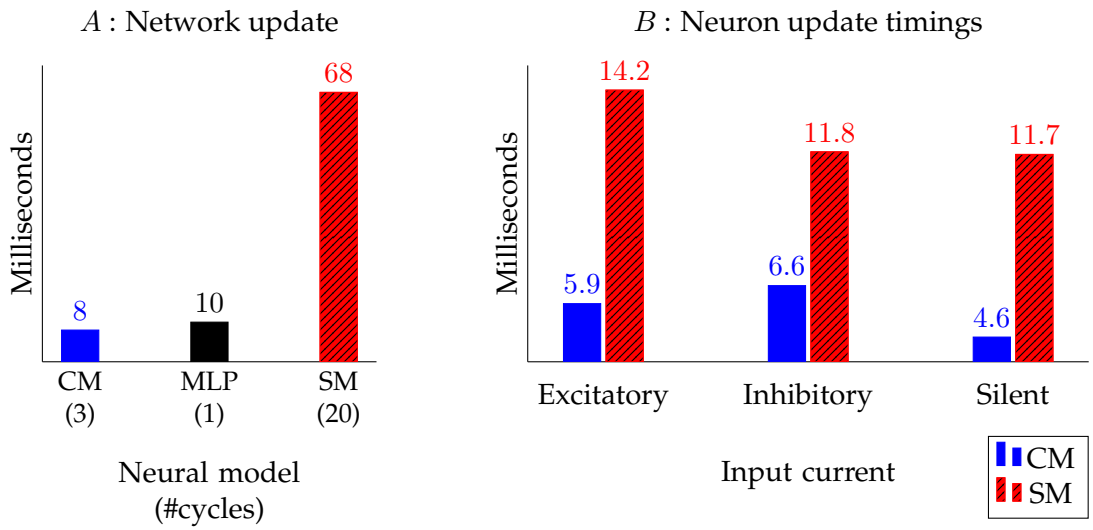


FIGURE 3.2: *A* shows the results from models each simulating 1000 randomized networks with 1000 hidden neurons each. *B* shows the results for the spiking models' average timings processing 1000 randomized neurons with 1000 updates each under different types of input

The benchmark results are depicted in figure 3.2. *A* shows the CM and MLP being roughly 7 times faster than the SM. The explanation for MLPs being on pair with the CM is because perceptrons will always propagate their data. Propagation is conditional for spiking neurons, and in our case results in fewer table lookups and overall calculations. It is worth mentioning that conditionals tend to make code slow, so results in general

[5]CPU: Intel(R) Core(TM) i7-3610QM @ 2.30ghz

depend on the number of propagations versus conditionals. Not seen by our data, the SM has a theoretical advantage per cycle given significantly lower spiking frequencies than the CM.

B shows the CM was on average 2.2 times faster than the SM at updating its neurons, with slight variation depending on conditions. We can see that the differences between the CM and SM are not as extreme when propagation and cycles are absent. The differences in timings based on conditional input currents show that some cause more stress than others. For example, idle CM neurons are significantly cheaper to process than active ones, which was not the case for SM. This is most likely due to special case CPU operations when zero is involved.

To summarize the benchmark results, we can see that the SM is very expensive to simulate compared to the other two modes. The SM's greatest toll is the amount of cycles necessary for a network update. Improving the model's neuron update efficiency, as done by other researchers[40], would most likely not improve overall timings in a very significant manner.

The expensive implementation of the SM is not a problem when dealing with few neurons, but it does become a real problem with larger as it scales linearly. Even though GPU implementations greatly boost the speed of MLP networks, the same is not always the case for SNNs [41] due to memory usage and conditionals. Also, using the GPU for these tasks would limit graphical rendering for ordinary desktop computers. This is especially unfitting if neural simulation were to be implemented for game clients.

Chapter 4

Testbed environment

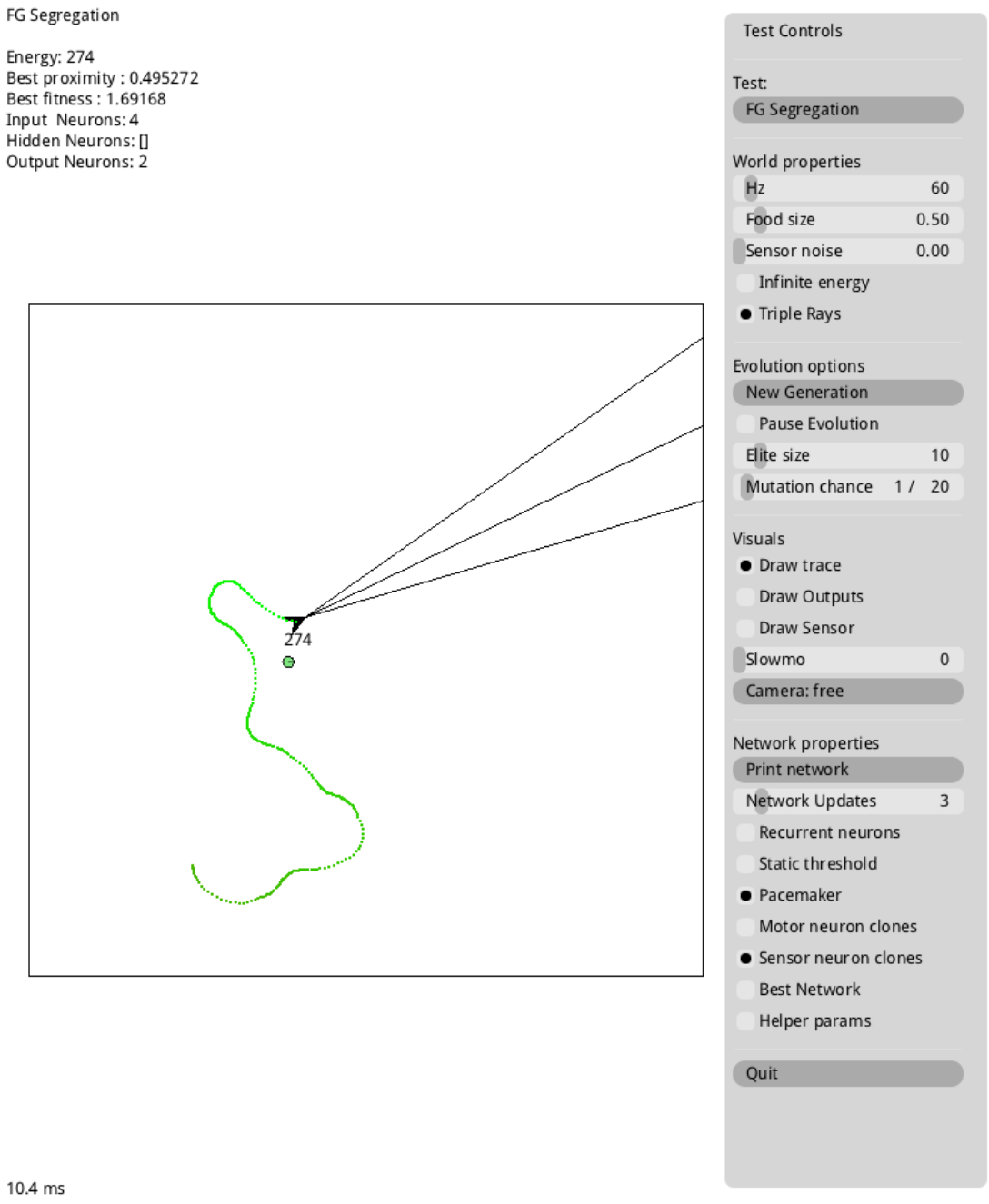


FIGURE 4.1: Testbed environment and GUI.

4.1 Overview

In order to test neural models for their behavioral capabilities, a testbed was created using *dbox*, a D programming language ported version of Box2D, a two-dimensional physics engine. This gives us the possibility of building a world in which a creature able to move and interact with objects. Several behavioral tests were designed, all of which sharing common mechanics to be specified in this chapter. However, only two tests will be highlighted in this thesis, with their specificity explained in the next chapter. Excluded tests are described in appendix D.

The main goal for a creature in any test is survival. It has a basic metabolism which burns energy, in addition to burning even more by performing motor actions. Collecting food restores energy, and it is up to the neural network to process information from the creature's sensors to direct its motors for navigation. Hence, the most successful creatures are those that collect food using minimal amount of energy. In order for us to train these neural networks based on how well they act as a controller, we will simulate evolution on a population of networks over several generations. We do this by subjecting each network to a test *trial*, injecting it into a creature and give it a limited amount of time to survive. For perspective, the network can be seen as a brain, while the creature is a body burning fuel.

4.1.1 The creature, food, and energy.

Any trial starts with the creature's energy pool set to a test specific reward value $E \leftarrow R$, and ends when $E = 0$. As the creature stands still or moves around in the world, a basic metabolic rate E_{meta} decrements energy together with overall motor activation. The *actuators* (motors) and their impact on the creature are shown in 4.2, and they are shared for all tests. Energy consumption for activating either of the two actuators is up to 5 times the basic metabolic rate, depending on their degree of activation [0:1]. The new energy value as a result of expenses is shown by:

$$E' \leftarrow E - (E_{meta} + 5E_{meta}(A_l + A_r))$$

As an example with E_{meta} being 1, fully activating A_l would result in an energy drain of 6. In order for be able to collect more energy, a *pickup zone* (Figure 4.2) of a fixed radius is attached to the center of the creature. Whenever this zone overlaps a food object, a successful *pickup* is said to happen. Pickups increments a counter p, followed by adding energy to the creature's pool. At any time, there is always one food object present in the world. When this food is picked up it will disappear to make another spawn at a different location. Given that we want every trial to be of finite time, a discount factor δ is set which ensures that the reward decrements exponentially per pickup. Pickup consequences are shown by:

$$\text{if } pickup \begin{cases} p' \leftarrow p + 1 \\ E' \leftarrow E + R \cdot \delta^{p'} \end{cases}$$

where $\delta = 0.8$ for all tests performed in this thesis.

For visuals, the creature's shape is formed like an arrow to indicate which direction it is facing. It also leaves behind a colored history trail, blending from green (most recent) to red. This makes it easy for the testbed user to track its movement and momentum throughout time. The creature also has a set of sensors S, which are defined in chapter 5 as they are test dependent.

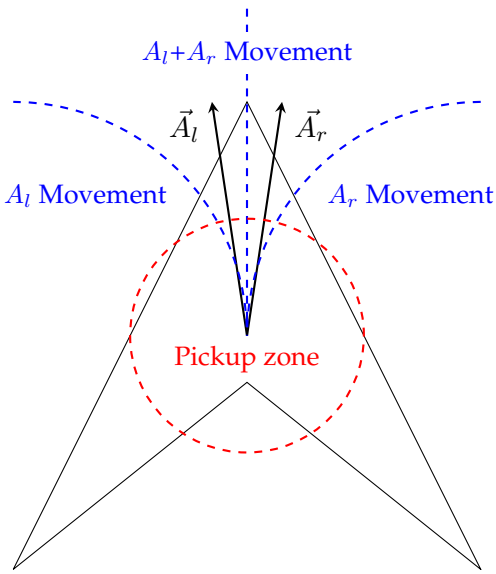


FIGURE 4.2: Creature anatomy. Food gets picked up when it is overlapping with the circular pickup zone. Arrows show force vectors applied to the rigid body (slightly below center of gravity) as a result of the actuators A_l and A_r which are activated by a neural network.

4.2 Evolutionary algorithm

As our intentions are to evolve a network capable of adapting to behavioral task with minimal knowledge of an optimal solution, we employ an evolutionary algorithm which is set to explore a multitude of solutions through a population of networks. Just like Darwinian evolution, we aim to increase the *fitness* of the population by recombining and mutating 'genes' between individuals in order to produce offspring. In our case, the genes would be the weights and parameters belonging to the neural model. These offspring networks are commonly referred to as *solutions*, as they evolve towards solving a problem under a specific context. In our testbed, the problem is maximizing fitness according to a fitness function in conjunction with the environment.

4.2.1 Fitness evaluation

In order to rank networks by performance, a simple fitness function is employed. As a trial ends as a result of zero energy, the network gets awarded with a score of pickups p in addition to a normalized value [0:1] reflecting the distance traveled from last pickup (or starting location) to the next. This extra distance-bonus is of great help in the start of a population, awarding networks that can at least move in the right direction.

4.2.2 Selection and Crossover

In order to combine genes, we select two parent networks based on fitness to produce an offspring. For our implementation, we use Stochastic Universal Sampling (SUS)[42] for selecting parents. The benefits with this method is that it respects the relative differences in fitness within the population, instead of just the ranks. This means that the best solutions of any generation will be the ones most probable to produce one or multiple offspring. As the parent networks are selected, a *crossover* occurs, meaning neurons (holding parameters and weights) mix at a uniform probability in order to construct an

offspring network. After the offspring is produced, there is a chance of mutation set to happen which will be described in the next section.

4.2.3 Mutation

Considering the networks in this thesis operates with fairly few neurons, the mutation rate is set to be 5% across all parameters. This is to keep the chance of duplicates in our population low. If a parameter is selected to mutate, it will undergo various forms of mutation operators. The CM model's operators are listed in 4.1, while the MLP borrows its weight mutation operator (including bias). For Izhikevich's SM network, the mutation operations are identical to those used in [9], with a few alterations based on preliminary testing. Specifically, the input is scaled to [0:20], weights range narrowed to [-50:50], while the delay parameter is discarded as connections bare no temporal delays. These alterations were done to make it perform and evolve better in our testbed.

Parameter	Gaussian	Rand [0:1]	Rand [-1:1]	setTo : 0	setTo : 1
mDecay, tDecay	50 %	25 %		12.5 %	12.5 %
equilibrium	50 %	25 %		25 %	
weight	50 %		25 %	25 %	

TABLE 4.1: Mutation probability distribution for the Controller Model. Chance for any parameter to mutate is set to 5% across all tests. If mutation occurs, one mutation operator will be selected stochastically to alter the parameter. Gaussian probability distribution σ = .05.

4.3 Neural Network integration

In order to isolate the neural model as the dependent factor for behavioral performance, all models operate with a fully connected FF architecture. Preliminary testing of models with recurrent and/or delayed connections did not show any significant benefits, and thus these features were excluded. The weights of connections in the SNN models signify the spike's strength as a fixed input value. The weights for MLP on the other hand is a coefficient on the input value traversing throughout the network, as shown in figure 2.4.

For the SNNs, an extra input neuron was given to act as a pacemaker, having a reflexive weight subjected to mutation. This ensures that the creature is still able to perform behavior independent of its sensors. This is comparable with the bias already embedded in MLP.

Specific model implementations were identical to those used in our earlier benchmarks (section 3.5.1).

4.3.1 Input / Output

For inputs, the MLP network operates according to figure 2.3. For the SNNs however, each of the creature's sensors/actuators have their own dedicated neuron. For neural identifiers, N_x resembles the neuron associated with S_x or A_x. During a network update, each sensor value is fed to its associated sensor neuron at every cycle.

The network's output decides the degree of activation of the creature's actuators. This is trivial for the perceptron, as it already outputs a graded value [0:1]. The SNNs however produce spikes, which can be interpreted as binary activations 0 and 1. For simplicity, A_x

activation is set to 1 if at least one spike occurred in the motor neuron during the of a network update. This gives a slight advantage to the MLP in terms of actuator control per timestep, as it can for example throttle the actuator at 50% force (and energy consumption). However, preliminary testing did not show this to be a significant factor for the functionality related to behavior. This might be explained by the SNNs ability to regulate the creature's speed by the frequency of spikes in the motor neurons. This is because inertia persists throughout the world's timesteps as we are moving a physical object (the creature).

4.4 Runtime cycle

Figure 4.3 gives an overview of the evolutionary cycle of our testbed. At every iteration of the main loop in our program, networks are tested in the same scenario, awarded a fitness based on their performance, and finally evolve into a new generation. Population size is set to 100 networks for all tests, and the top 10% is considered the *elite*. The elite gets a free pass to join the next generation, which are retested in the next scenario. This way we can keep an aggressive mutation rate without risking to lose the best solutions.

A striking feature of our testbed is its simulation speed. With the help of CPU[1] multi-threaded worlds, simulation time per generation ranged from 50ms to 500ms. This range is due to the trained networks (creatures) surviving longer, hence getting more simulation time. This high speed greatly facilitated preliminary testing, as the author could get quick feedback while tweaking variables.

[1]CPU: Intel(R) Core(TM) i7-3610QM @ 2.30ghz

```
/* Initialize an array of networks of a set size, and
   randomize their parameters                                    */
population ← blankPop(populationSize);
population ← randomize(population);

/* Start main training loop                                      */
while true do
    /* Randomize scenario parameters to make every trial
       unique.  (Spawning locations etc.)                        */
    S ← createScenario();

    foreach network in population do
        world ← newWorld(network, S);
        runTest(world);
    end

    population ← sortByFitness(population);
    elite ← getBestSolutions(population, eliteSize);

    /* Initialize an array for offspring                         */
    offspring ← blankPop(population.length − eliteSize);

    /* Create new networks from the old generation               */
    foreach network in offspring do
        parents ← getParents(population);
        network ← makeChild(parents);
        network ← mutate(network);
    end

    /* Replace old gen with new gen                              */
    population ← offspring + elite;
end
```

FIGURE 4.3: Pseudo-code for the evolutionary algorithm within the testbed. Blue text are arrays of networks.

Chapter 5

Behavioral experiments

In this chapter we propose two tests which aim to measure the performance of three different neural models as controllers. The first model is the MLP, acting mostly as a control due to being time-tested. The second and third are of the spiking kind, namely Izhikevich's SM and lastly our CM. The SM was chosen due to its richness (NCFs), as it forms the basis of the CM. In addition, the SM has been shown to be functional for other behavioral implementations [8, 9]. The LIF model was also considered, but was ultimately excluded as preliminary tests did not show any promising results.

5.1 Temporal Edge Detection

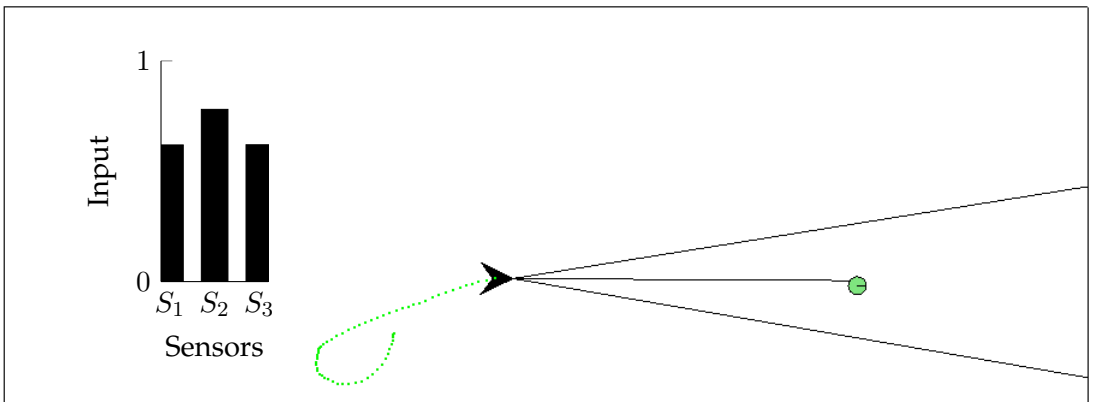


FIGURE 5.1: Creature input for the TED test with triple-rays. The figure shows the upper part of the square box with the creature in the center. The black rays return a normalized distance between the creature and the first object (wall or food).

An original behavioral test was designed with spiking neural networks in mind, dubbed the Temporal Edge Detection (TED) test, shown in figure 5.1.

For sensors, the creature is equipped with a frontal ray that provide feedback regarding the distance of the first object it touches. To form an analogy, the 'eye' can thus only see the distance in form of intensity on a min-max scale, much like a color spectrum.

We hypothesize that the creature can only detect the edge of food in relation to a background wall, as there is a sudden peak in sensor intensity during a ray sweep. Considering the MLP is stateless, it is not expected to perform this task as it cannot compare the last state with the next. Therefore, a variation of this test include additional rays (triple) as sensors, offering a different form of edge detection which is independent of states (non-temporal). This is possible by merely comparing the input of one ray relative to another.

The creature and food are spawned within a square box, the box having sides (walls) of length s. In order to prevent spawning inside the walls, and to not make it too difficult to detect food up against them, the spawn zone is set to be 80% of the original square box originating from the center. The ray sensor value is set to $1-\frac{d}{d_{max}}$ where d is the Euclidean distance between the hit object and the creature, and d_{max} is $\sqrt{2} \cdot s$. This normalizes the sensor to a range of [0:1] depending on proximity of the hit object. A sensor value of 1 is up close, while 0 is out of reach. For triple-rays, the additional rays had their angle set to 3° left and right of the center ray. In order to train faster and eliminate possible exploitation of the triple-ray scenario, the SNNs sensory neurons have shared parameters. If not, one of these extra neurons could evolve to be non-adaptive and only respond to avoid bumping into walls, which is an unintended advantage.

Neural architecture for the SNNs is set to an input neuron per sensor (ray) in addition to our pacemaker, fully connected to our two motor neurons. For the MLP, the only difference is that each sensor data is fed to all the hidden neurons. The number of hidden neurons was equal to the amount of sensors.

For any model to be qualified as functional for behavior, the requirement is for its solution to collect more than 2 food on average during its lifetime. The food's size is equal to the creature's pickup radius.

5.1.1 Results

Table 5.1 shows the results for the TED test, while supporting graphs are shown in figure 5.2. The CM's final solutions were on average significantly better than the SM (unpaired two-tailed t-test $p = .003$). Results with error-bars are shown in appendix B.1, while a typical behavioral strategy by the CM is shown in appendix C.1.

The spiking neural networks both evolved a technique which consisted of biased circling turning driven by the pacemaker, causing the ray to scan across walls. At moments with a sudden deviation in the sensor's intensity, such as the ray hitting food, the sensory neuron fires to cause a slight counter-turn. This caused a feedback loop in which the creature jitters its ray between the food and the background wall while moving forward. The MLP network was only functional for triple-rays, where its behavior consisted of biased turning and moving towards food dead center.

Network	Rays	Mean	σ	Best	Functional?
CM	1	6.31	2.13	9	✓
SM	1	5.72	2.21	7.7	✓
MLP	1	1.08	0.06	1.2	
CM	3	5.08	1.74	8	✓
SM	3	5.42	2.77	8.2	✓
MLP	3	4.26	0.96	5.7	✓

TABLE 5.1: TED results

5.1.2 Discussion

As hypothesized, the MLP network was unable to evolve any functional behavior for the one-ray test. This is due to its absence of stored temporal data, something the spiking neurons do within the membrane. It is only when the MLP utilizes triple-rays that it is

able to find the food, underpinning a non-TED strategy. Although not tested, double-rays are also believed to be sufficient performing the same function.

The spiking neural networks were both successful at detecting and moving to food. For both the one-ray and triple-ray tests, they had a strong preference for sticking to the same behavior of performing jitters. With our methods, it is hard to verify if the SNNs were able to perform non-TED strategies considering evolving TED was faster with one-ray. After all, the architecture of the one-ray test can be a subset of the triple-rays network. This because the weights from additional sensor neurons (rays) could evolve to be zero. If redundant sensors are present, we might not be surprised seeing the faster learning rate for the one-ray SNNs compared to triple-rays. This is because fewer weights need to be tuned.

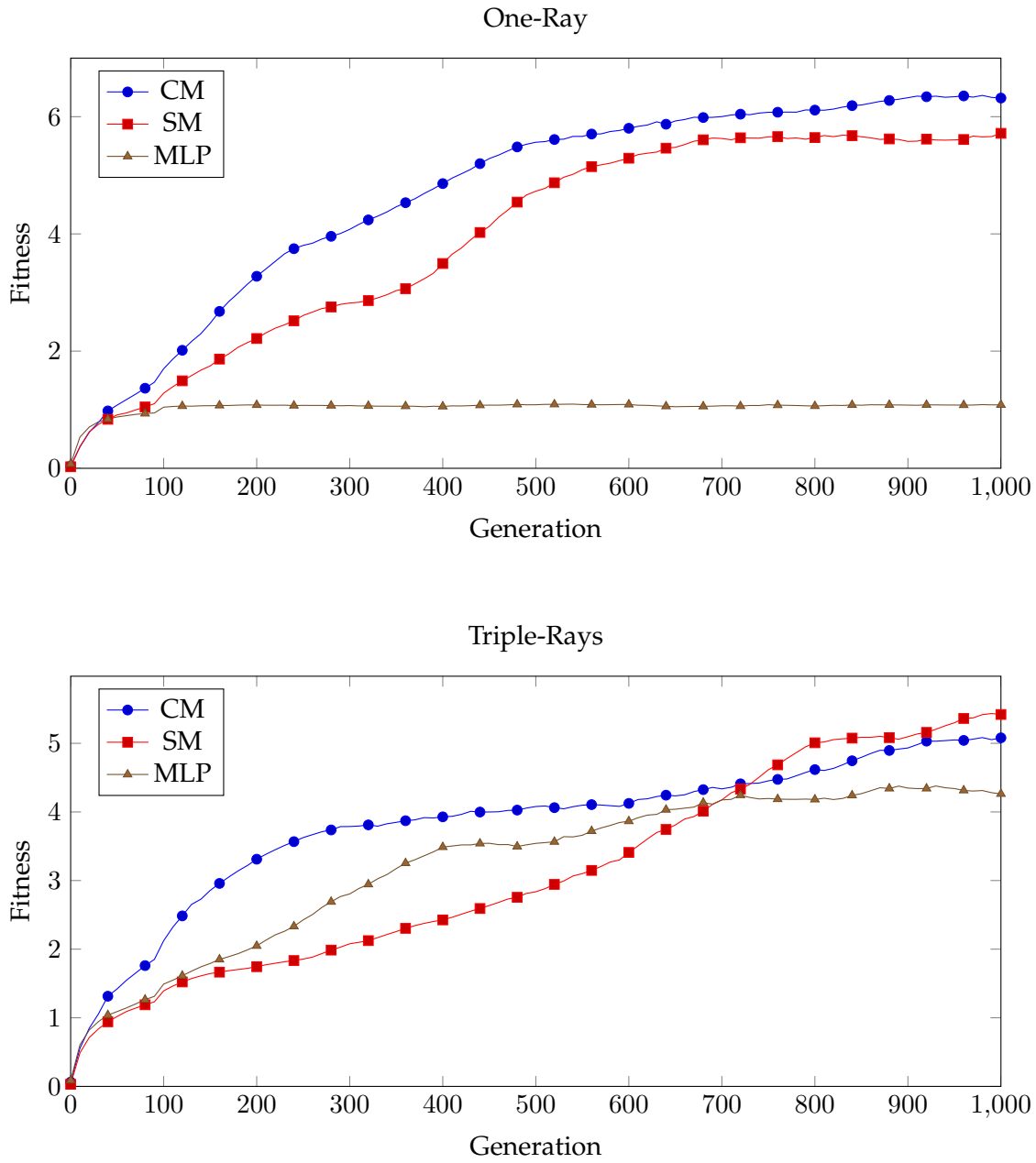


FIGURE 5.2: Results from the TED test showing different neural network models training 20 populations each. Graph shows the running mean of the best solutions over the last 100 generations.

5.2 Chemotaxi

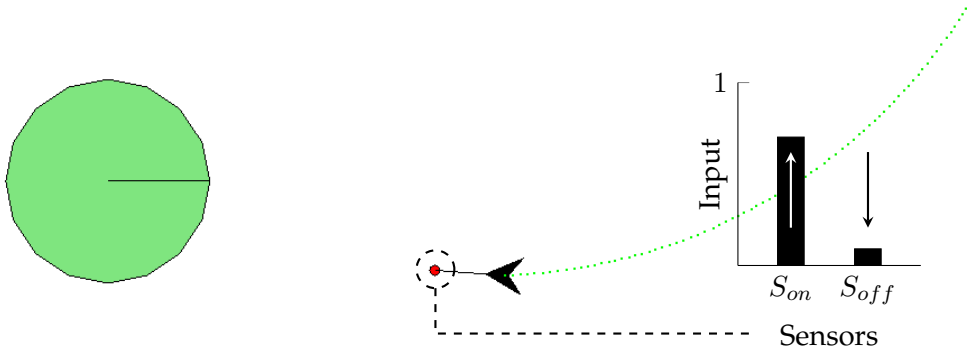


FIGURE 5.3: Creature input for the 'Chemotaxi' test. As the creature's elongated nose (marked by a dashed circle) approaches the large food on the left, S_{on} increases while S_{off} decreases.

A very common behavior of animals - especially invertebrates - involves navigating towards the source of chemicals. The chemotaxi test described here aims to assert our creature's ability to perform the same task by being able to smell the food. The method used here is inspired by the work of Izquierdo & Lockery [43], who were able to produce a minimalistic neural circuit of the *c.elegans* performing chemotaxis. In their methods, they used different artificial neurons operating under GPs, which is similar to *c.elegans's* neurons as they do not produce APs[21]. Another paper[44] also showed the possibility of using LIF neurons responding to GP from specialized sensory neurons.

Our creature starts at $spawn_0$, where every $spawn_n$ is pre-generated to be positioned at a random distance between 0 and d_{max} from $spawn_{n-1}$ at a random angle. d_{max} is a fixed value which is fine tuned together with the energy reward for pickup. Associated with each spawn, there is also a coefficient value α within a randomized range [0:1]. This value is related to the intensity of the food's chemical emission. This forms a neatly chained proximal path of spawns, ensuring that the creature has a decent chance reaching them before energy runs out.

Similar to [43], we will use two of *c.elengans's* sensory neurons that activate to either the increase or decrease in salt concentration from the environment. Our sensors are thus labeled S_{on} and S_{off}, the names representing on or off target (salty food in our case). The sensor values are

$$S_{on} = \alpha(1 - \frac{d}{d_{max} \cdot 1.1})$$
$$S_{off} = 1 - S_{on}$$

where d is the Euclidean distance between the creature's *nose* and food. The nose is located at the tip of an elongated antenna sticking out the front of the creature, shown in figure 5.3. The fixed coefficient value 1.1 is to ensure that the sensor range is 10% larger than d_{max}. This is to provide the creature with some sensory leeway under the unlucky scenario where food spawns close to d_{max}. In the case where $d > d_{max}$, S_{on} = 0 (out of range). Much like the TED test, the SNNs sensory neurons are set to have shared parameters due to their similar nature of opposite behavior.

Although odor concentration is usually of Gaussian distribution, leaving creatures to train in such an environment can lead to unintended exploitative strategies. This is

because the concentration value relates to the proximity of the source, as pointed out by Izquierdo & Lockery [43]. To control for this, we borrow their technique in our own methods. This technique includes the randomized α coefficient, which alters the max concentration intensity at the center of the food. As a result the creature agnostic to the proximity of the food source, something it could have exploited otherwise. For example, preliminary testing showed that it sometimes used one of the sensory neurons to produce a counter-turn whenever the sensor reached a specific value.

To assert whether the creature has generalized the behavior of always moving towards higher concentration agnostic of distance, the best trained network will be cherry-picked for a sub-test. This sub-test involves the creature being exposed to a food source emitting odor of a Gaussian distribution, instead of a linear one.

Considering this Chemotaxi experiment is a close replication to the work of Izquierdo & Lockery, the most important differences will be made explicit in this paragraph. While our sensors output raw distance to the neurons, theirs activated based on derivative operator measuring the difference in concentration within an interval. The motor neurons, modeled by simple first order nonlinear dynamics, also received inputs from both an oscillating central pattern generator to facilitate *undulations* - which is to move in a snake-like fashion. In fact, their creature was enforced to perform undulations as a requirement for realistic movement. This restriction was not active in our test, as the author was curious whether or not undulation was able to naturally evolve as a chemotaxi strategy. In order to facilitate this, our creature has its nose (holding the sensors) at the tip of an extended antenna. Undulation would thus cause the antenna to sweep left and right to sample the environment, which could emulate the head of the creature. Lastly, while their trial consisted of moving to one food only, ours follow the testbed environment where a new food spawns when the former is picked up. This offers us the chance to observe whether or not the creature has problems adapting to a sudden change in sensor value after pickup.

Neural architecture for the SNNs are set to include one sensory neuron per sensor (S_{on} and S_{off}) in addition to our dedicated pacemaker for the input layer. The hidden layer consists of two neurons, followed by one motor neuron per actuator (A_l and A_r). MLP differs by feeding the sensors to an extra first hidden layer of 2 neurons. The fully connected FF architecture is similar to *c.elegans* as it has been shown to have very little recurrent connections [24]. However, in *c.elegans* the hidden neurons are questionably redundant given that they seem to merely forward information rather than process it [11]. Our method will include them none the less, just in case some models might put them into use. A sub-test seeks to explore the minimal amount of neurons required to perform chemotaxis for our creature, stripped of the hidden layer. The dependent variables for this test are the various combinations of a pacemaker, S_{on} and S_{off} sensors.

For any model to be qualified as functional for behavior, the requirement is for its solution to collect more than 2 food on average during its lifetime. The food's size is 10 times the creature's pickup radius.

5.2.1 Results

Table 5.2 shows the results for the chemotaxi test, while supporting graphs are shown in figure 5.4. Results with error-bars are shown in appendix B.3, while typical behavioral strategies by the CM is shown in appendix C.2. For naturalistic Gaussian based concentration levels, reliable behavior of the CM is shown in figure 5.5.

Both the ON and OFF neurons evolved significantly better with a supporting PM neuron (paired two tailed t-test $p = .008$ and $p = .018$ respectively). Several solutions

Model	Mean	σ	Best	Functional?
CM	8.66	2.42	11.5	✓
SM	1	0.05	1.1	
MLP	0.72	0.05	0.8	
SM*	8.96	1.73	11	✓

Neurons	Mean	σ	Best	Functional?
ON+PM	9.40	1.70	12.5	✓
ON	8.21	2.58	12.3	✓
OFF+PM	5.97	2.26	10.5	✓
OFF	4.93	2.07	9.5	✓

TABLE 5.2: Results for the 'Chemotaxi' test. Lower table shows the CM using different input neuron combinations.

by the CM produced undulating behavior. The evolved creatures showed no issues re-adjusting to the sudden new input value post-pickup.

Undulation was successfully achieved using a minimalistic network compromised of only 3 neurons (Sensor neuron N_{on} and motor neurons N_l and N_r). Behavior of N_{on} is shown in figure 5.6. N_l received a weak weight from N_{on}, causing it to spike tonically (A). This resulted in a biased forward-turn. N_r received a stronger inhibitory weight from N_{on}, causing it to release a rebound burst (N) once N_{on}'s activation halted full frequency spiking. This gave N_r the role of counter-steering whenever it was off course.

5.2.2 Discussion

Although it was expected that the MLP network would be dysfunctional, the same cannot be said for the SM. In order to investigate further, a custom SM network (SM*) was made where the input neurons were CM neurons. Given that this custom network was functional, this indicates that the SM has issues with adaptive generator potentials. More precisely, it has problems with macro-adaption opposed to the micro-adaption required in the TED test. Different forms of input scaling was also attempted under the suspicion that our default input range might be too artificial for the more biological-based model. The most likely explanation for SM's failure could be rooted in the fact that it is intended for modeling cortical spiking neurons, not sensory neurons of graded potentials. Another interesting find was that SM* never showed any solutions with undulations. This might be explained by the different NCFs available in the motor neurons. The rebound bursts made by the CM can be extremely long, even when it is compressed over 3 cycles.

Unlike the TED test which always ended up with one strategy, the various solutions for the chemotaxi test were quite diverse. The best solutions produced undulating behavior, where the creature continuously sweeps its sensor left and right to sample the environment. There were also other reliable strategies (as shown in appendix C.2), however energy inefficient. Several of these involved biased turning with pirouettes, which is coincidently very common for *c.elegans*[45]. From observations, it was quite clear that once the population evolved into this form of strategy, it was nearly impossible to mutate into an undulating one. This is because the solutions reach a *local maxima*, where small mutations will not be able to produce radically different behavior. This is quite common

problem in learning algorithms with multiple peaks in the solution space. Considering there is no competition in our environment, these sub-optimal solutions will never be wiped out by a better population.

Since only the CM qualified as being functional, it was also the only one tested for minimal circuitry. Here we find the absolute best solutions of chemotaxi behavior for our testbed environment, showing a significant benefit when the network is facilitated with a pacemaker neuron. However, this does not mean that the pacemaker was required for optimal behavior, seeing as the best solutions had very similar fitness (12.5 and 12.3). The results also showed that the S_{on} sensor was undoubtedly better compared to the S_{off} neuron. This is most likely due to the S_{off} neuron being mostly dormant when the creature approaches food, barely producing any spikes at all to elicit behavior. This might explain why we see a facilitating pacemaker evolving faster, along with a better final solution for the OFF+PM condition (10.5 over 9.5). If the pacemaker is able to produce a biased forward movement, the S_{off} only has to signal when the creature is off course.

The discrepancy in efficiency between S_{on} and S_{off} in these results are the complete opposite of the findings in [43]. However, there are many reasons to explain this contrast, the main one being that the implementations are fundamentally different. In their version, sensory neurons assisted to correct motor neurons which received an oscillating input from a central pattern generator [46]. Discussed in their paper, the N_{off} neuron is better at correcting wrong trajectories when the creature is already moving forward.

As described in our methods, the best evolved creature (a CM network) was placed in a custom scenario for illustrative purposes (shown in figure 5.5) with a Gaussian concentration distribution. The results confirmed its evolved strategy to always move up the gradient, independent of the non-linear increase of input. The creature made biased trajectories to come in from one side of the food. Other solutions did display a more direct trajectory using wider undulations, but they were less efficient. Interestingly, increasing the length of our creature's antenna resulted in a more direct path. This was because the sensors underwent bigger fluctuation during the sweeps caused by the undulation, resulting into the motor neurons to adjust earlier. In contrast, making it shorter produced wider undulations as a result of the sweeps sampling less spacial data.

As explained in a paper[44], sensory neurons in *c.elegans* generates a heavy wave of GP when the concentration increases or decreases, only settling to a resting state when there is no change. The CM seems to be able to emulate this neural behavior by going straight from input to adaptive APs, independent of a GP sensory neuron. This is indicative by the resulting figures 5.5 and 5.6, giving insight to how well the CM neuron interprets rise and fall of concentration levels, even at extremely small numerical perturbations.

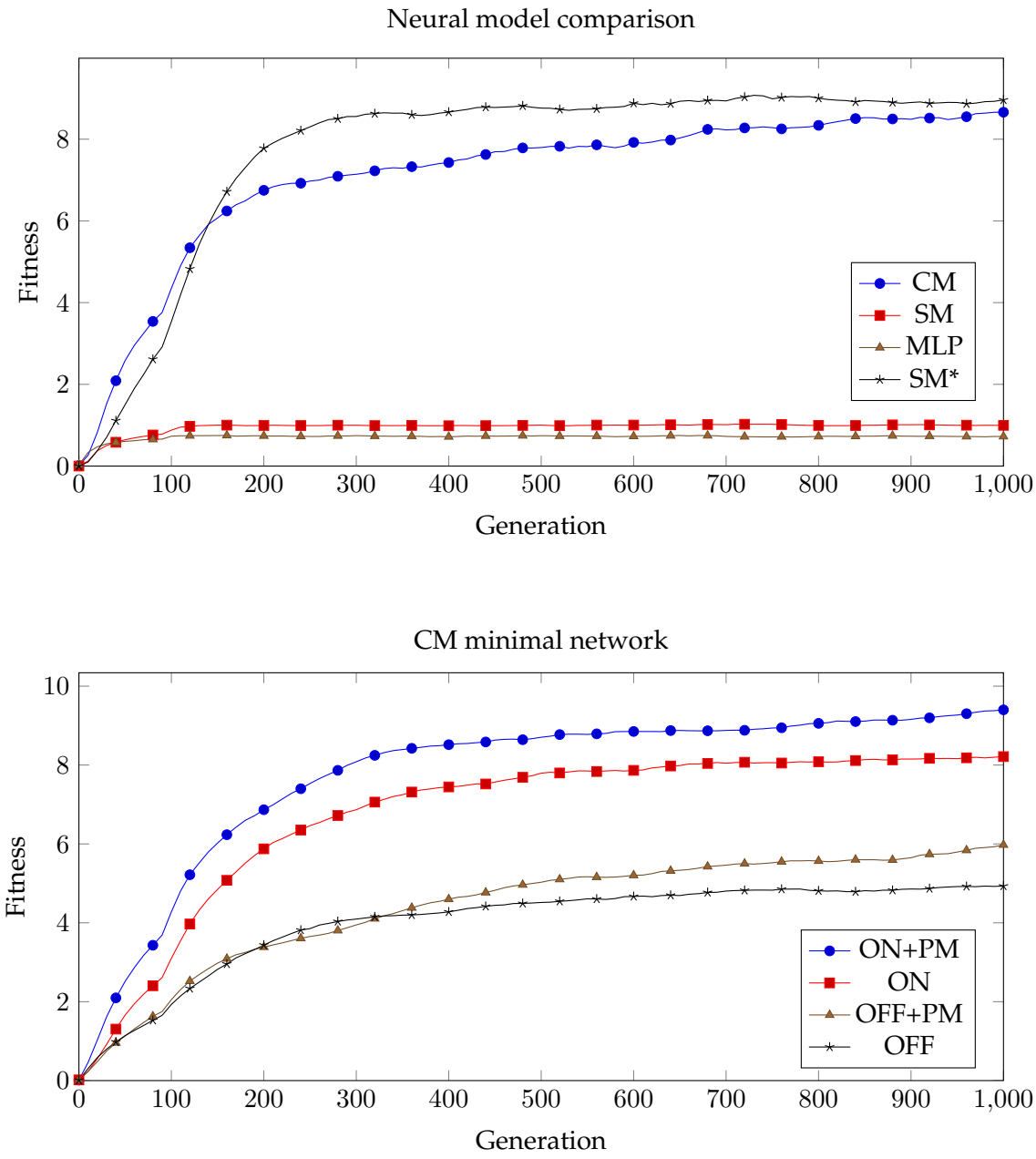


FIGURE 5.4: Results from the 'Chemotaxi' test, showing different neural network models training 20 (50 for minimal network) independent populations each. Graph shows the running mean of the best solutions over the last 100 generations. SM* is assisted with CM input neurons.

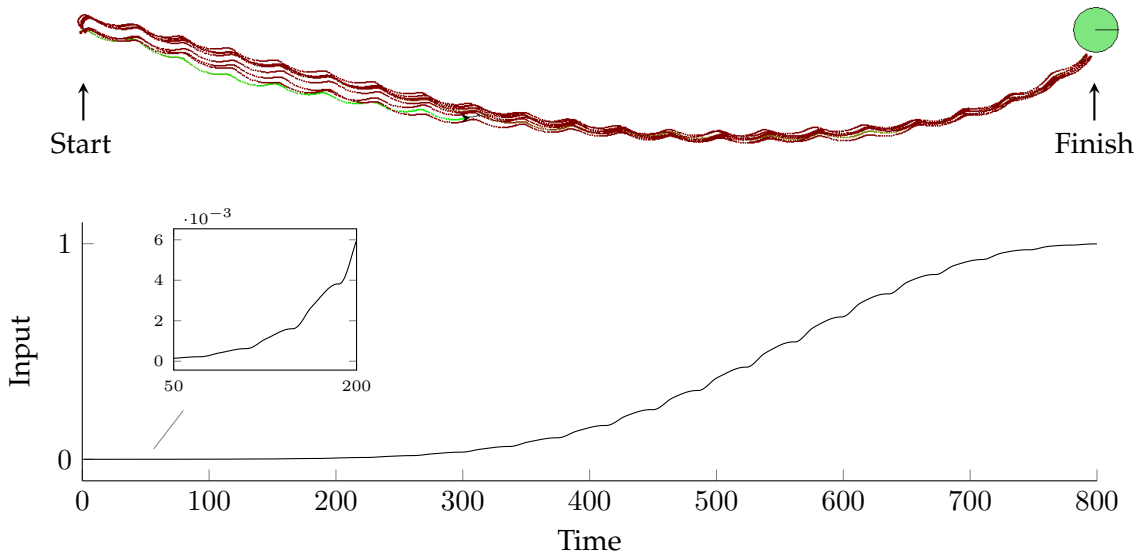


FIGURE 5.5: Chemotaxi behavior for Gaussian concentration levels. Upper image shows multiple paths taken by the same creature (CM) given random spawn orientations at "start" navigating towards finish. Lower graph shows S_{on} as a result of its movement, including an enhanced segment of the earlier timesteps. Pickup reliability was 100% for 1000 trials limited to 1000 timesteps.

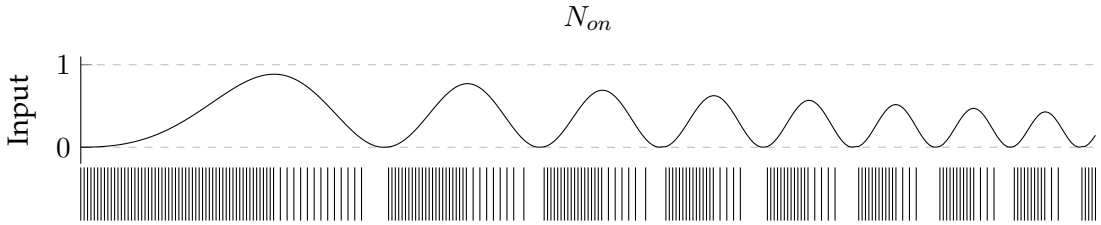


FIGURE 5.6: The evolved input neuron N_{on} for the CM network shown in figure 5.5. An artificial sinusoidal input is fed to the neuron, showing the discrete output spikes as vertical bars below. The neuron elicit full spiking frequency whenever the input increases, while dropping to a lower frequency otherwise. $N_{para} = \{a = 0.33, b = 1, c = 0\}$.

Chapter 6

Evaluation and Conclusions

6.1 Evaluation by requirements

As mentioned in the introduction, the CM will be evaluated according to the requirements listed in section 1.1.

The first requirement, richness (1), is evaluated in section 3.4 by comparing it to a list of NCFs. The CM was far richer than the LIF model, ranking very close to the SM.

The second requirement, cheapness (2), is evaluated in section 3.5. In the case of typical network implementations, the CM was up to 7 times faster compared to the SM, being on par with the MLP. Also in the case for individual neuron updates, the CM was 2.2 times faster than the SM.

The final requirement, functionality (3), is evaluated based on the behavioral experiments in chapter 5. For the TED test, the CM was on par with the SM. For the chemotaxi test however, CM was the only network that could perform.

6.2 Future research and possible applications

As the CM is new, the author hopes it might be further developed. For example, a desirable trait for a spiking neural model is to have a variable timestep. The fact that our CM does not support this, means that a trained network could behave differently if we were to alter the physics step.

Related to the topic of timesteps is the concept of network cycles. The relation between behavioral performance and the amount of cycles in the CM network update is not fully understood, as it was originally intended to operate with just one cycle. Further investigation of this topic might reveal interesting insight of network mechanics.

Shifting our focus to networks, preliminary tests did not show any advantage in performance based on architecture other than the FF kind. However, network architecture is an interesting topic in general, as connections are rather sparse within biological brains[47, 48]. What behavior cannot FF networks perform? Interestingly, one of the excluded tests of this thesis, listed in appendix D.1, was impossible for the creature to achieve without recurrent connections.

As for evolution, one might try to improve the learning speed by building the network of a fixed amount of neuron types that share parameters. Not only would this be more realistic in terms of biological neurons, but it would also reduce the evolutionary search space, leaving the main focus on mutating weights. This might reveal classes of neurons which are useful as a starting point for building any network.

In some papers[4, 5], researchers evolve the creature's morphology (bodies and sensor/actuator locations) alongside the neural network. Implementing the CM for these types of creatures might render it able to utilize morphology for a behavior which was unsupported by another neural model. This behavior (following the axiom in 3) could require a specific NCF in which only our CM would be capable of.

As we have shown the CM's capability to adapt to a wide range of numerical values in the chemotaxi test, this might be useful for evolving a controller for a virtual car. In racing games, the CM might utilize temporal dynamics of ray sensors reading the proximity of nearby walls or cars. This could be used for indicating impending collisions where cars up front suddenly breaks. Although [8, 9] made comparisons between the SM and MLP for racing games, they did not get significant results showing the SM to be better. Since our result show that the SM has problems with certain types of adaption, this raises potential benefits using our CM.

Although for our implementation we associated one neuron for each sensor, future implementations should consider giving it to multiple. For example, if one was interested in registering a set threshold (H) and acceleration of a sensor value (S_{on} in 5.6), one neuron would not be enough.

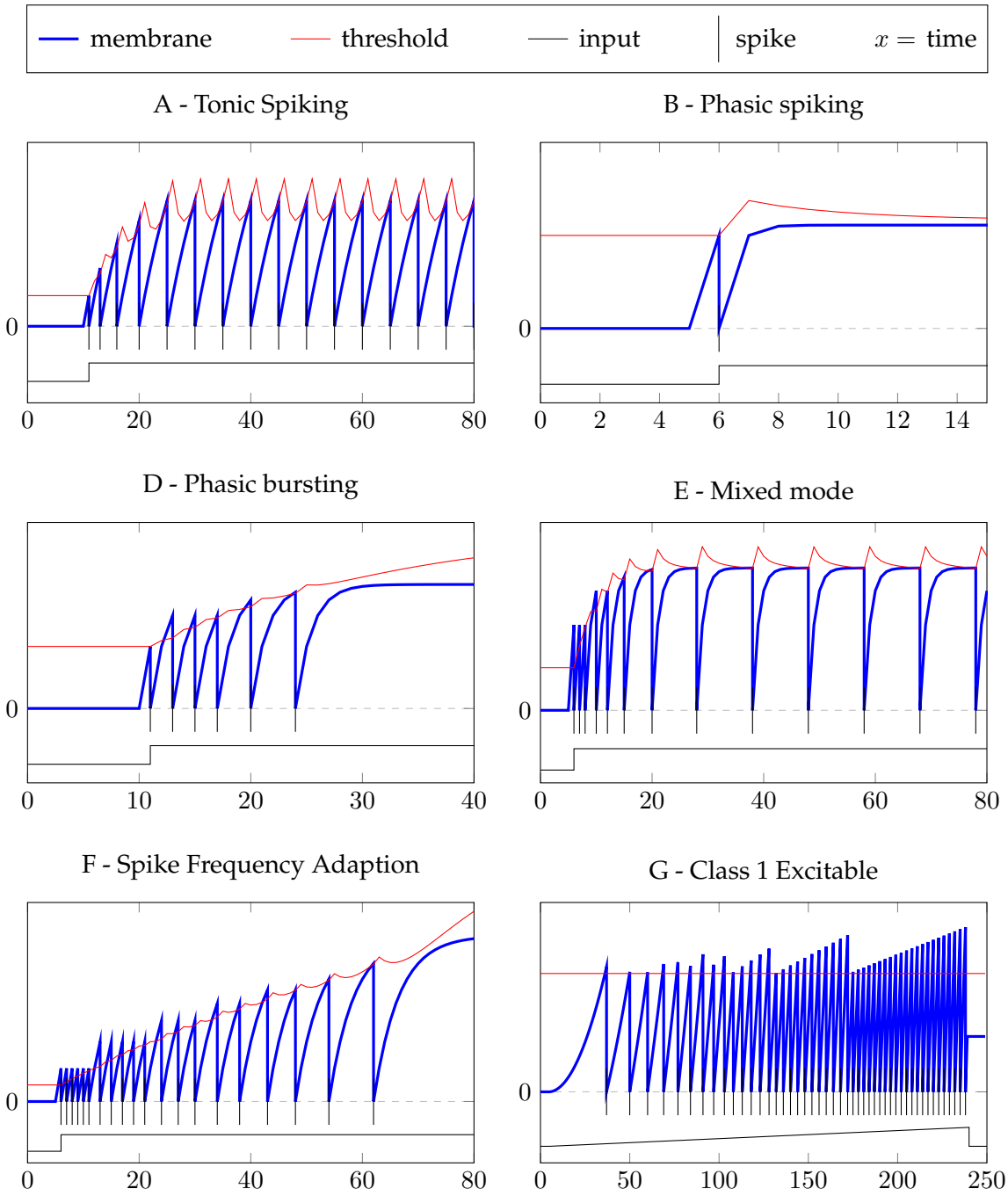
Finally, it is worth mentioning that the CM has been used at MeinMein[49], where the author did his internship. Here, the CM was successfully used for evolved controllers in creatures simulated in 3D physics environments.

6.3 Conclusion

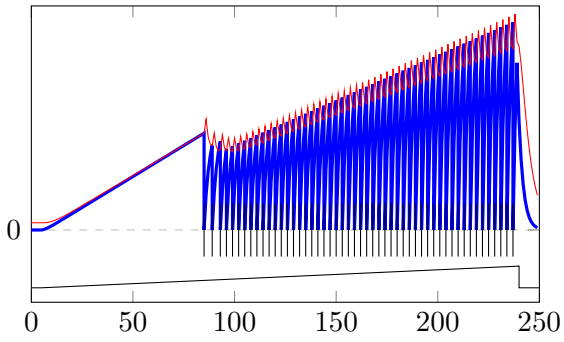
In this thesis, we have explored the capabilities of a new spiking neural model created as a controller for virtual environments. We have shown that it is able to replicate most of NCFs of biological neurons, while also being cheap to implement for runtime environments with large timesteps. For its evolved functionality, it is equivalent to the SM for the TED test, while being a the only model that could show chemotaxi behavior. It was identified that the SM is restricted regarding adaption to wider numerical ranges of input current. The CM had such problem, and was in turn able to evolve different chemotaxi strategies using only 3 neurons. This network was also able to perform undulating behavior independent of a central pattern generator. We also showed that a pacemaker neuron with constant input may facilitate both functionality and learning speed of the CM. For both behavioral tests done in this paper, it is clear that the SNNs have an advantage over the more traditional MLP due to temporal dynamics of neuron states.

Appendix A

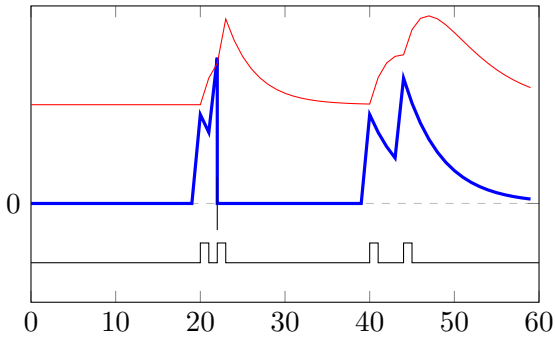
Neuro-computational features of the Controller Model



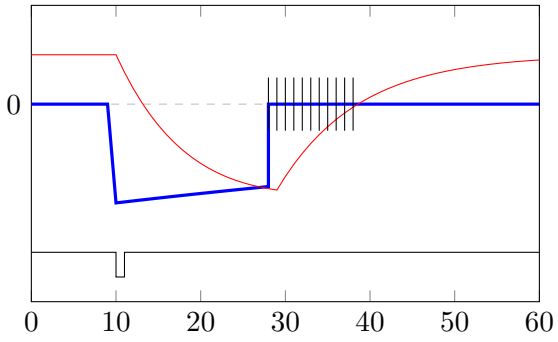
H - Class 2 Excitable



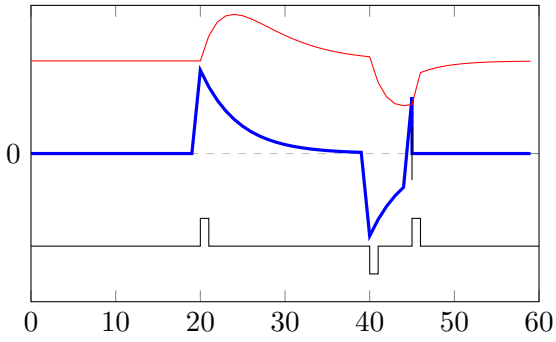
L - Integrator



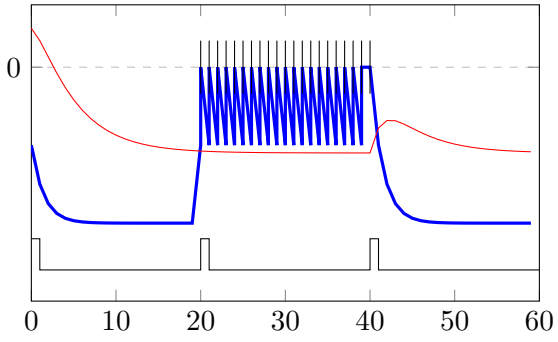
N - Rebound burst



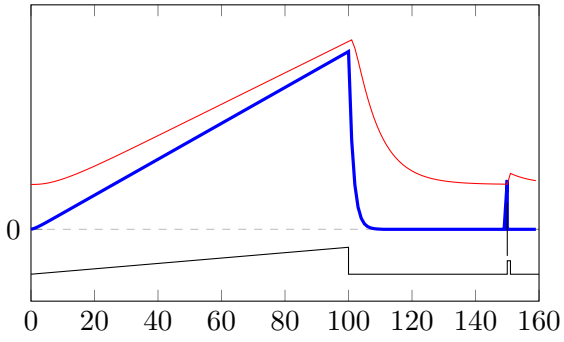
O - Threshold Variability



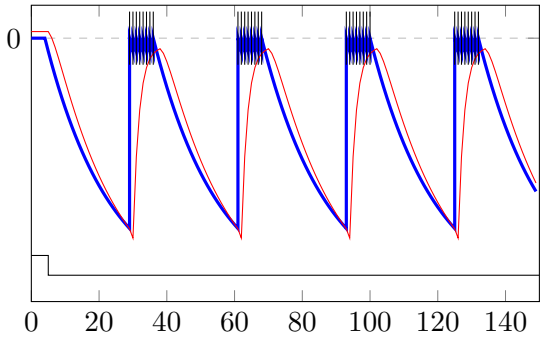
P - Bistability



R - Accomodation



T - Inhibition Induced Bursting



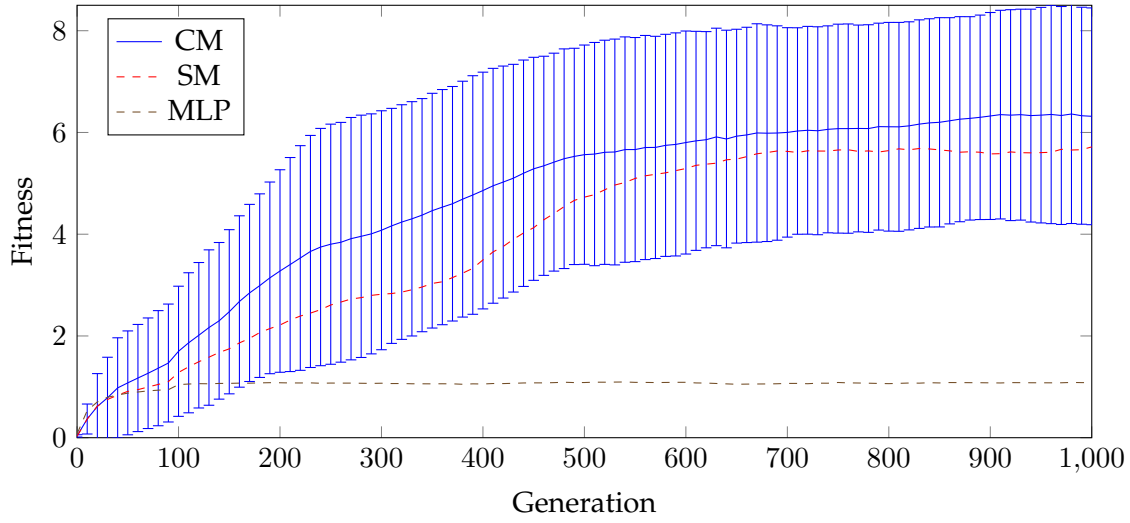
NCF	a	b	Input
A	0.9	0.8	+0.5
B	0.1	0.5	+0.5
C	-	-	-
D	0.5	0.1	+0.5
E	0.4	0.3	+1
F	0.8	0.1	+1
G	1	0	+0.003t
H	0.65	0.5	+0.03t
I	-	-	-
J	-	-	-
K	-	-	-
L	0.8	0.5	+0.45 : t[20,22,40,44]
M	-	-	-
N	0.99	0.2	-1
O	0.8	0.5	+0.45 : t[20,45], -0.45 : t[40]
P	0.5	0.4	-1, +1 : t[40,80]
Q	-	-	-
R	0.5	0.25	+0.01t : t[0:100], +0.55 : t[150]
S	-	-	-
T	0.95	0.85	-1

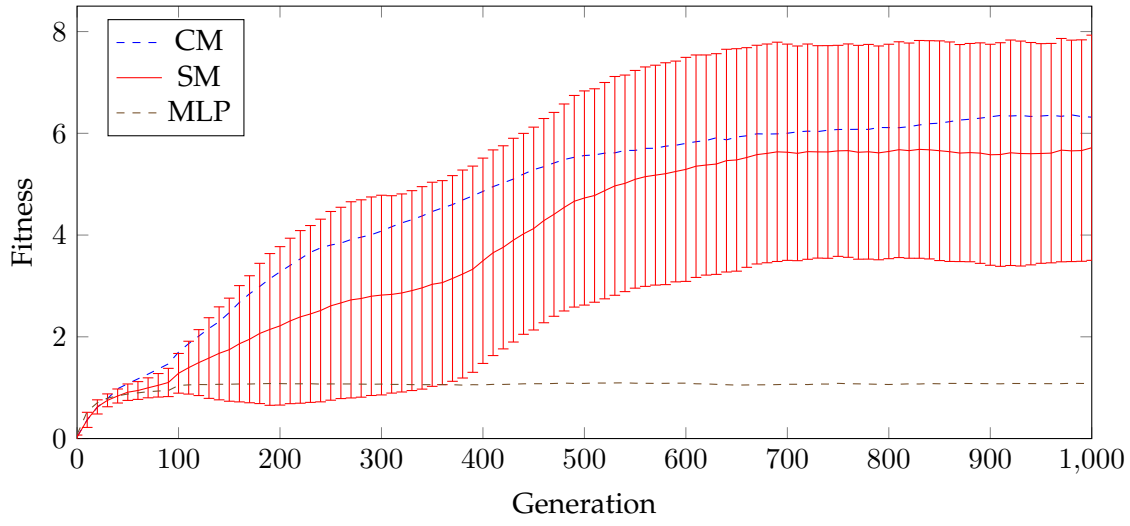
TABLE A.1: CM parameter settings for replication of neuro-computational features. c = 0.5 for all cases.

Appendix B

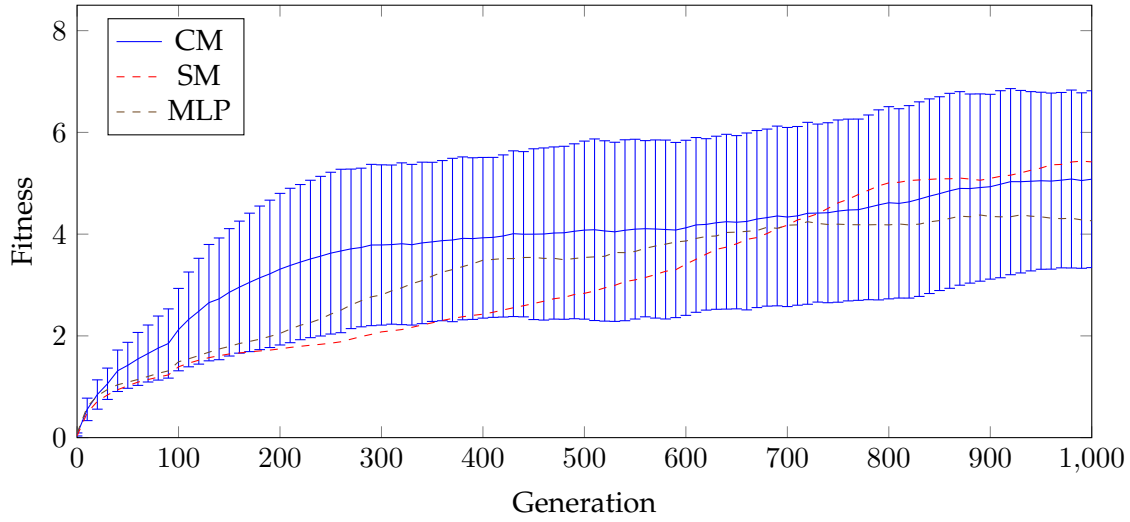
Test results with error bars

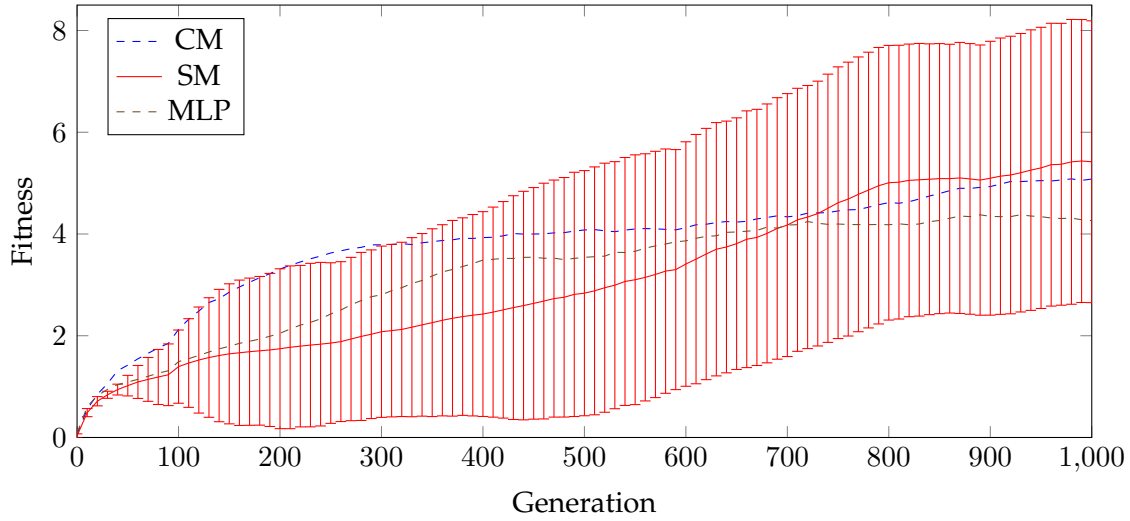
B.1 Temporal Edge Detection : one-ray

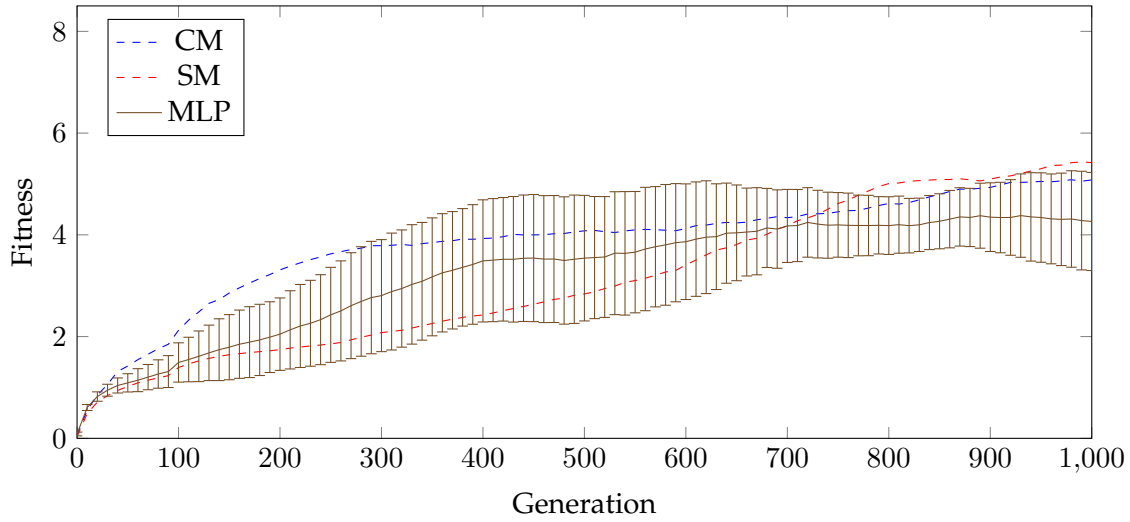




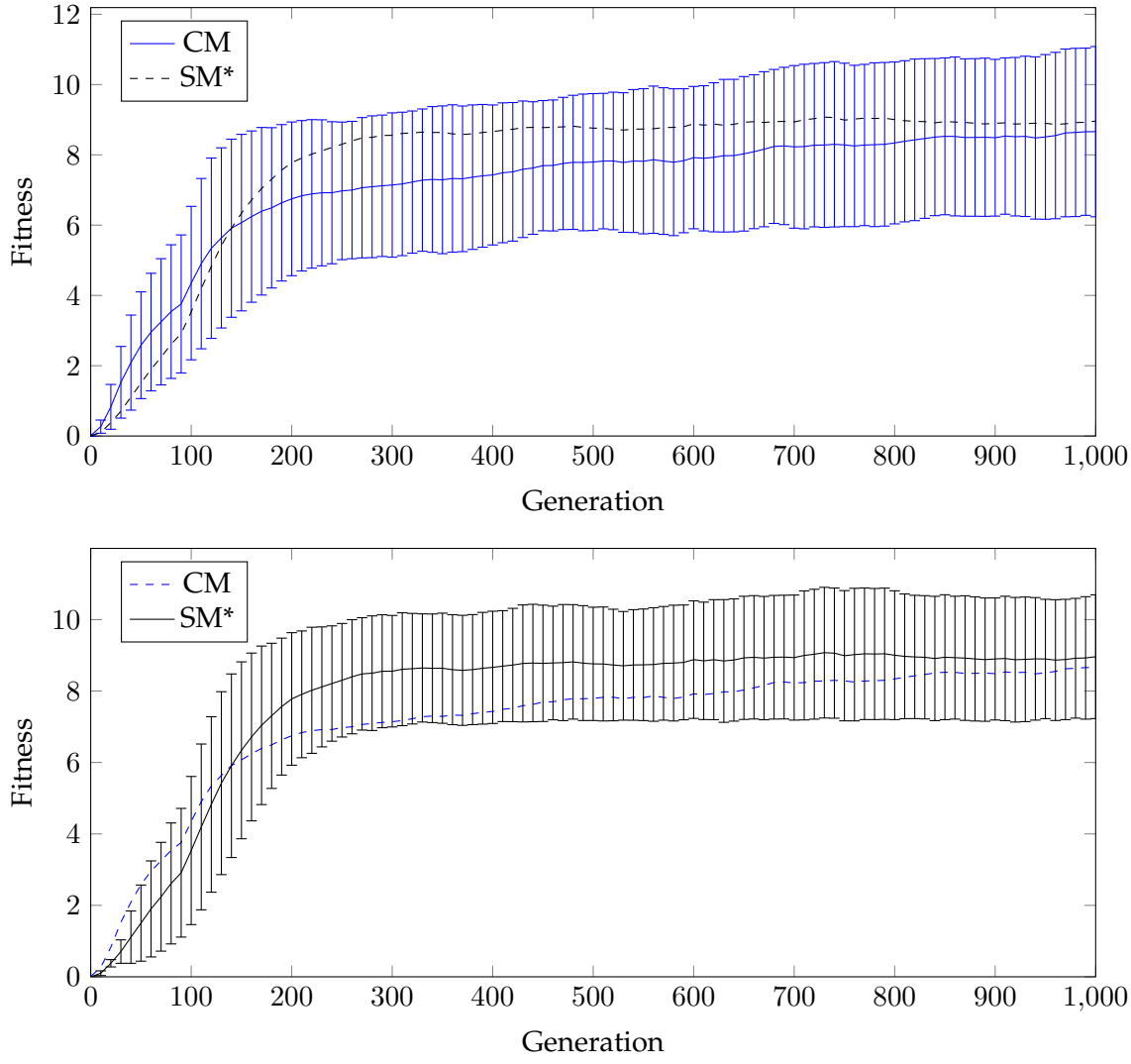
B.2 Temporal Edge Detection : triple-rays







B.3 Chemotaxi



SM* is assisted with CM input neurons.

Appendix C

Evolved creature behavior using the Controller Model

C.1 Temporal Edge Detection

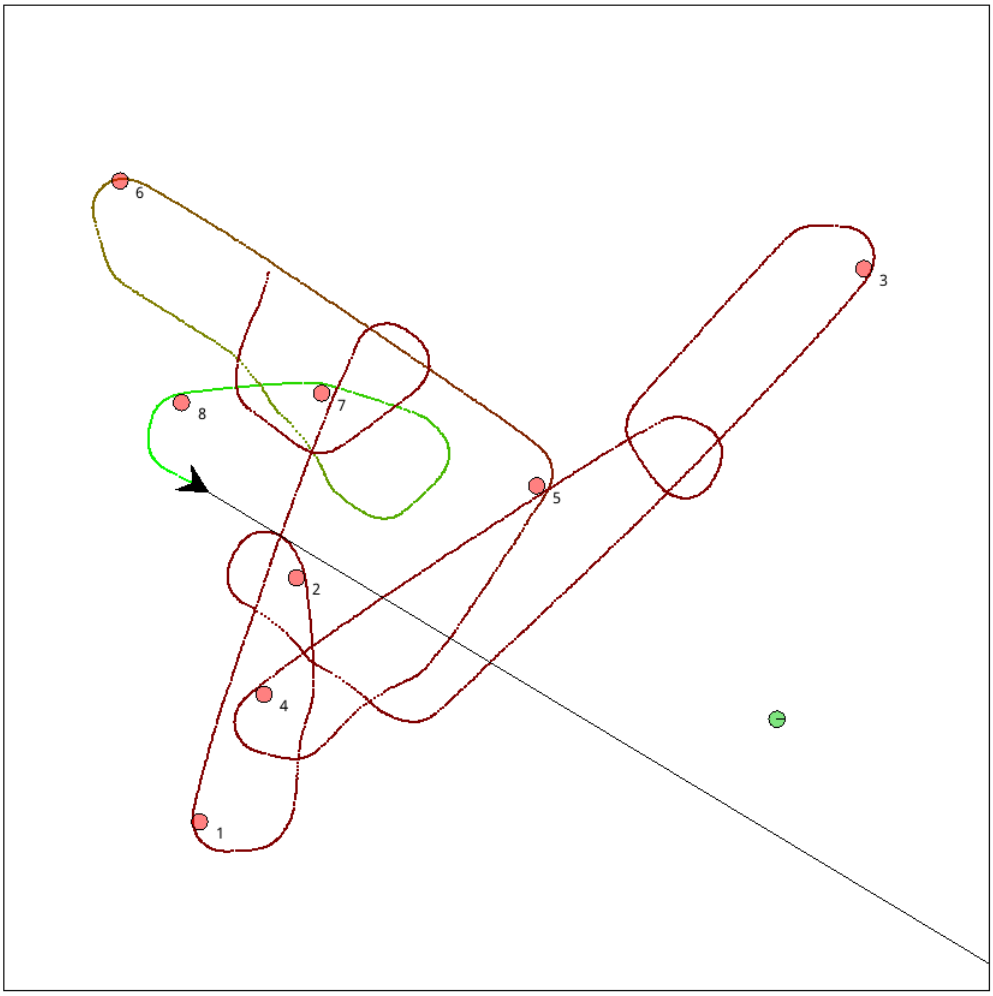


FIGURE C.1: Temporal Edge Detection (one-ray) strategy. The behavior of triple-rays was more or less identical.

C.2 Chemotaxi

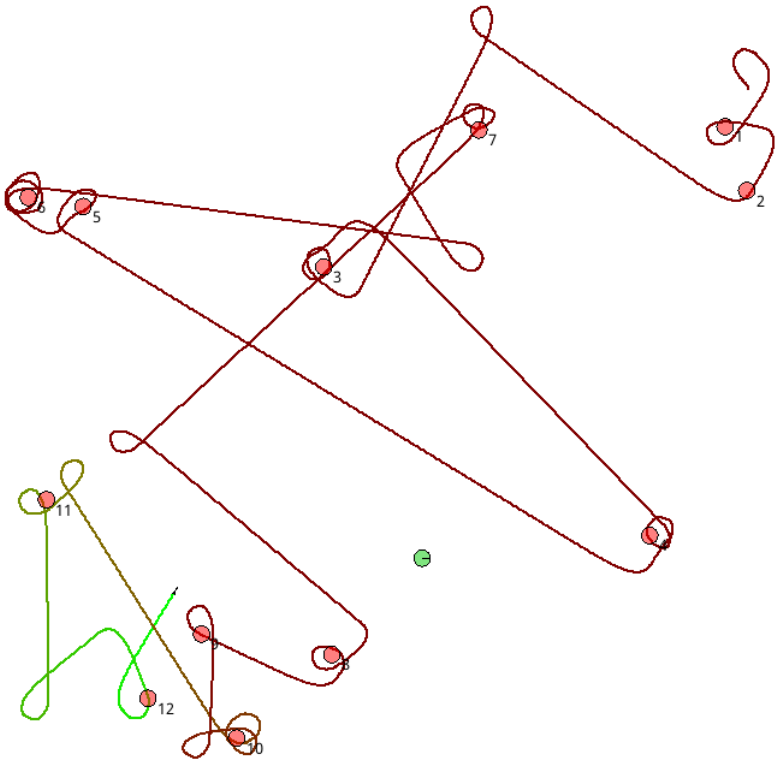


FIGURE C.2: Chemotaxi strategy - Sharp turns

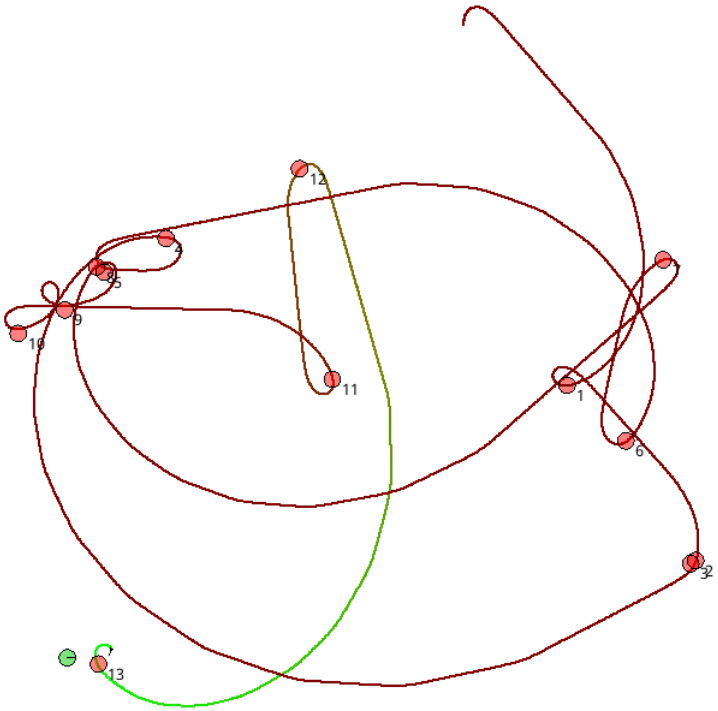


FIGURE C.3: Chemotaxi strategy - Gradual turns

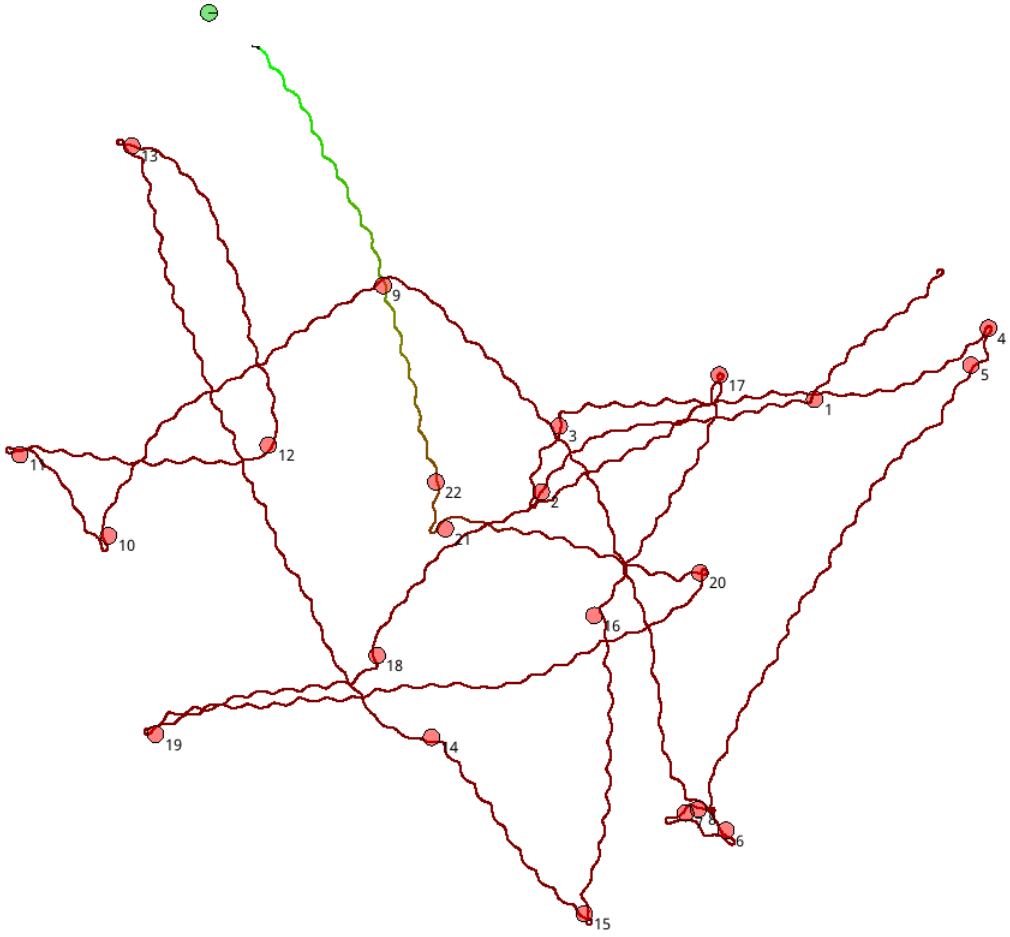


FIGURE C.4: Chemotaxi strategy - Undulation

Appendix D

Excluded tests

This appendix includes tests that were excluded from this thesis due to scope limitation.

D.1 Tracker

The 'Tracker' tests involved an immobile creature staring at a moving ball with limited peripheral vision. The task was for the creature to turn in the direction it last saw the ball (on its left or right). This was the only test in which feedback connection were essential, where connections from the motor neurons went back to the hidden layer. This made the network able to switch internal states of its turning bias, triggered by sensory neurons (rays). This means that if the creature last saw the ball on its left side, it would keep on turning left forever, vice versa. Solutions for this problem was successfully evolved for all tested models.

D.2 Blind Pathfinder

The 'Blind Pathfinder' aimed at evolving a fixed action pattern, independent of any sensory input. Here a predetermined race track was made, challenging the creature to navigate through it "by heart". This test was merely intended to explore sequential coding schemes of SNNs, where the CM vastly outperformed the SM. Using 100 hidden neurons fueled by a single pacemaker of constant input, it was possible for the network to learn tracks with multiple sharp turns.

D.2.1 Junction

An extended version of the blind pathfinder was the 'Junction' test. Here there was one main track which eventually branched into two, yet only one of these tracks was currently active. Before arriving at the branching junction, a conditional input spike was fed to the creature when it moved over a specific location. This gave the creature a signal of which track was active, making it possible for the network to evolve internal branching. As a result, when fully trained it was able to finish either of the tracks based on the scenario, implying both sequences were stored inside the same neural network. This was most likely possible due to the bistability NCF (P). Although the SM might be able to perform the same after enough brute force training, the author never witnessed it being able to pass the junction. This might be related to how input is fed as a constant current over the course of network cycles.

Bibliography

[1] Quoc V Le. "Building high-level features using large scale unsupervised learning". In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 8595–8598. DOI: 10.1109/ICASSP.2013.6639343.

[2] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489. DOI: 10.1038/nature16961.

[3] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint* 1312.5602 (2013). URL: https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf.

[4] Karl Sims. "Evolving 3D morphology and behavior by competition". In: *Artificial life* 1.4 (1994), pp. 353–372. URL: http://www.karlsims.com/papers/siggraph94.pdf.

[5] Dan Lessin, Don Fussell, and Risto Miikkulainen. "Adapting morphology to multiple tasks in evolved virtual creatures". In: *ALIFE 14: The Fourteenth Conference on the Synthesis and Simulation of Living Systems* 14 (2014), pp. 247–254. URL: http://nn.cs.utexas.edu/?lessin:alife14.

[6] Stephen Grand, Dave Cliff, and Anil Malhotra. "Creatures: Artificial life autonomous software agents for home entertainment". In: *Autonomous Agents and Multi-Agent Systems* 1.1 (1998), pp. 39–57. URL: http://link.springer.com/article/10.1023/A%3A1010042522104.

[7] Mänttäri and Joonatan. "Applications of Artificial Neural Networks in Games; An Overview". In: *IRCSE '11: IDT Student Workshop in Research Methods* Mälardalen University, Sweden (2011). URL: http://www.idt.mdh.se/kurser/ct3340/ht11/MINICONFERENCE/FinalPapers/ircse11_submission_5.pdf.

[8] Elias Yee and Jason Teo. "Evolutionary spiking neural networks as racing car controllers". In: *International Journal of Computer Information Systems and Industrial Management Applications* ISSN 2150-7988 Volume 5 (2013), pp. 365–372. DOI: 10.1109/HIS.2011.6122141.

[9] Urszula Markowska-Kaczmar and Mateusz Koldowski. "Spiking neural network vs multilayer perceptron: who is the winner in the racing car computer game". In: *Soft Computing* 19.12 (2015), pp. 3465–3478. DOI: 10.1007/s00500-014-1515-2.

[10] T. Yamazaki and J. Igarashi. "Realtime cerebellum: A large-scale spiking network model of the cerebellum that runs in realtime using a graphics processing unit". In: *Neural Networks* 47.0 (2013), 103—111. URL: http://www.sciencedirect.com/science/article/pii/S0893608013000348.

[11] J. G. White, E. Southgate, J. N. Thomson, and S. Brenner. "The structure of the nervous system of the nematode caenorhabditis elegans". In: *Philosophical Transactions of the Royal Society of London. B, Biological Sciences* 314.1165 (1986), 1—340. DOI: 10.1098/rstb.1986.0056.

[12] W.R. Schafer. "Deciphering the neural and molecular mechanisms of C. elegans behaviour". In: *Current Biology* 15.17 (2005), R723–729. DOI: 10.1016/j.cub.2005.08.020.

[13] Brett Szymik. "A Nervous Journey". In: *ASU - Ask A Biologist* (2011). URL: http://askabiologist.asu.edu/neuron-anatomy (visited on 03/25/2016).

[14] Jonathan Platkiewicz and Romain Brette. "A threshold equation for action potential initiation". In: *PLoS Comput Biol* 6.7 (2010), e1000850. DOI: 10.1371/journal.pcbi.1000850.

[15] Unknown. 2014. URL: http://www.physiologyweb.com/lecture_notes/neuronal_action_potential/neuronal_action_potential_frequency_coding_in_the_nervous_system.html.

[16] JGR Jefferys and HL Haas. "Synchronized bursting of CA1 hippocampal pyramidal cells in the absence of synaptic transmission". In: *Nature* 300 (1982), pp. 448–450. DOI: 10.1038/300448a0.

[17] Jan-Marino Ramirez and Andrew K Tryba and Fernando Peña. "Pacemaker neurons and neuronal networks: an integrative view". In: *Current opinion in neurobiology* 14.6 (2004), pp. 665–674. URL: http://www.ncbi.nlm.nih.gov/pubmed/15582367.

[18] Eugene M. Izhikevich. "Which model to use for cortical spiking neurons?" In: *IEEE transactions on neural networks* 15.5 (2004), pp. 1063–1070. URL: http://www.izhikevich.org/publications/whichmod.pdf.

[19] Ellen Covey. n.d. URL: http://courses.washington.edu/psych333/handouts/coursepack/ch13-Information_processing_in_retina.pdf.

[20] Svirskis G, Gutman A, and Hounsgaard J. "Electrotonic structure of motoneurons in the spinal cord of the turtle: inferences for the mechanisms of bistability". In: *J Neurophysiol.* 85.1 (2001), pp. 391–398. URL: https://www.ncbi.nlm.nih.gov/pubmed/11152739.

[21] Lockery S, Goodman M, and Faumont S. "First report of action potentials in a C. elegans neuron is premature". In: *Nature neuroscience* 12.4 (2009), pp. 365–366. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3951996/.

[22] Biswa Sengupta, Simon Barry Laughlin, and Jeremy Edward Niven. "Consequences of converting graded to action potentials upon neural information coding and energy efficiency". In: *PLOS Comput Biol* 10.1 (2014). DOI: 10.1371/journal.pcbi.1003439.

[23] Lars Buesing and Wolfgang Maass. "A spiking neuron as information bottleneck". In: *Neural computation* 22.8 (2010), pp. 1961–1992. URL: http://www.gatsby.ucl.ac.uk/~lars/papers/NeCo_BuesingETAL2010b.pdf.

[24] Jifeng Qian, Arend Hintze, and Christoph Adami. "Colored Motifs Reveal Computational Building Blocks in the C. elegans Brain". In: *PLoS ONE* 6.3 (2011), pp. 1–8. DOI: 10.1371/journal.pone.0017013.

[25] Frank Rosenblatt. "The perceptron: a theory of statistical separability in cognitive systems (Project Para)". In: *Cornell Aeronautical Laboratory* (1958).

[26] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.3 (1988). DOI: 10.1038/323533a0.

[27] Alan L Hodgkin and Andrew F Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of physiology* 117.4 (1952), pp. 500–544. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1392413/.

[28] Richard B Stein. "Some models of neuronal variability". In: *Biophysical journal* 7.1 (1967), p. 37. DOI: 10.1016/S0006-3495(67)86574-3.

[29] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002. URL: http://icwww.epfl.ch/~gerstner/SPNM/node26.html.

[30] Romain Brette. "What is the most realistic single-compartment model of spike initiation?" In: *PLoS Comput Biol* 11.4 (2015), e1004114. DOI: 10.1371/journal.pcbi.1004114.

[31] W. Maass. "Networks of spiking neurons: The third generation of neural network models". In: *Neural Networks* 10.9 (1997), 1659—1671. DOI: 10.1016/S0893-6080(97)00011-7.

[32] Simon Thorpe and Arnaud Delorme and Rufin Van Rullen. "Spike-based strategies for rapid processing". In: *Neural networks* 14.6 (2001), pp. 715–725. URL: http://www.ncbi.nlm.nih.gov/pubmed/11665765.

[33] Romain Brette. "Philosophy of the Spike: Rate-Based vs. Spike-Based Theories of the Brain". In: *Frontiers in systems neuroscience* 9 (2015). DOI: 10.3389/fnsys.2015.00151.

[34] Corinne Beurrier et al. "Subthalamic nucleus neurons switch from single-spike activity to burst-firing mode". In: *The Journal of neuroscience* 19.2 (1999), pp. 599–609. URL: http://www.ncbi.nlm.nih.gov/pubmed/9880580.

[35] Eunji Cheong et al. "Tuning thalamic firing modes via simultaneous modulation of T-and L-type Ca2+ channels controls pain sensory gating in the thalamus". In: *The Journal of Neuroscience* 28.49 (2008), pp. 13331–13340.

[36] Eugene M. Izhikevich. "Resonate-and-fire neurons". In: *Neural networks* 6.14 (2001), pp. 883–894. URL: http://izhikevich.org/publications/resfire.pdf.

[37] David LaBerge and Ray Kasevich. "The cognitive significance of resonating neurons in the cerebral cortex". In: *Consciousness and cognition* 22.4 (2013), pp. 1523–1550. DOI: 10.1016/j.concog.2013.10.004.

[38] Yann A et al LeCun. "Efficient backprop". In: *Neural networks: Tricks of the trade* (1998), pp. 9–48. URL: http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf.

[39] Eugene M. Izhikevich. "Simple model of spiking neurons". In: *IEEE Trans. Neural Networks* 14 (2003), 1569—1572. URL: http://www.izhikevich.org/publications/spikes.pdf.

[40] Arash Ahmadi and Mark Zwolinski. "A modified Izhikevich model for circuit implementation of spiking neural networks". In: *First IEEE Latin American Symposium on Circuits and Systems (LASCAS)* (2010), pp. 192–195. DOI: 10.1109/LASCAS.2010.7410243.

[41] Romain Brette and Dan FM Goodman. "Simulating spiking neural networks on GPU". In: *Network: Computation in Neural Systems* (2012). URL: http://romainbrette.fr/WordPress3/wp-content/uploads/2014/06/BretteGoodman2012.pdf.

[42] James E. Baker. "Reducing Bias and Inefficiency in the Selection Algorithm". In: *Proceedings of the Second International Conference on Genetic Algorithms and their Application* (1987), pp. 14–21.

[43] Izquierdo E.J. and Lockery S.R. "Evolution and Analysis of Minimal Neural Circuits for Klinotaxis in Caenorhabditis elegans". In: *The Journal of neuroscience : the official journal of the Society for Neuroscience* 30.39 (2010), pp. 12908–12917. DOI: 10.1523/JNEUROSCI.2606-10.2010.

[44] Shibani Santurkar and Bipin Rajendran. "A neural circuit for navigation inspired by C. elegans Chemotaxis". In: *arXiv* 1410.7881 [cs.NE] (2014). URL: http://arxiv.org/pdf/1410.7881v1.pdf.

[45] Jonathan T Pierce-Shimomura, Michael Dores, and Shawn R Lockery. "Analysis of the effects of turning bias on chemotaxis in C. elegans". In: *Journal of Experimental Biology* 208.24 (2005), pp. 4727–4733. DOI: 10.1242/jeb.01933.

[46] Eve et al. "Invertebrate central pattern generation moves along". In: *Current Biology* 15.17 (2005), R685–R699. URL: http://www.ncbi.nlm.nih.gov/pubmed/16139202.

[47] Song et al. "Highly nonrandom features of synaptic connectivity in local cortical circuits". In: *PLoS Biol* 3.3 (2005), e68. URL: http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.0030068.

[48] Nicolas Brunel. "Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons". In: *Journal of computational neuroscience* 8.3 (2000), pp. 183–208. URL: https://galton.uchicago.edu/~nbrunel/pdfs/brunel00JCNS.pdf.

[49] Meindert Kamphuis. *MeinMein*. 2016. URL: www.meinmein.com.