UNIVERSITY OF CALIFORNIA
Los Angeles

Visual Tracking with Spiking Neural Networks in an Oculomotor Controller for a Biomimetic Model of the Eye

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Computer Science

by

Taasin Saquib

2022

ABSTRACT OF THE THESIS

Visual Tracking with Spiking Neural Networks in an Oculomotor Controller for a Biomimetic Model of the Eye

by

Taasin Saquib
Master of Science in Computer Science
University of California, Los Angeles, 2022
Professor Demetri Terzopoulos, Chair

Spiking neural networks (SNNs) are comprised of artificial neurons that, like their biological counterparts, communicate via electrical spikes. SNNs have been hailed as the next wave of deep learning as they promise low latency and low power consumption when run on neuromorphic hardware. Current deep neural network models for computer vision often require power-hungry GPUs to train and run, making them great candidates to replace with SNNs. In this thesis, we develop and train an SNN-based foveation deep neural network that enables a biomechanical model of the human eye to track a moving visual target. Inspired by the ON and OFF bipolar cells of the retina, we use event-based data flow in the SNN to direct the necessary extraocular muscle-driven eye movements. While many SNNs are conversions of trained deep neural network architectures, we train our SNN models from scratch using modified deep learning techniques. Classification tasks are straightforward to implement with SNNs and have received the most research attention, but visual tracking is a regression task. We use surrogate gradients and introduce a linear layer to convert membrane voltages from the final spiking layer into the desired outputs. Our SNN foveation network is noisier than the previously used deep neural network, but it enhances the biomimetic properties of our virtual eye model and enables it to perform reliable visual tracking.

The thesis of Taasin Saquib is approved.

Sriram Sankararaman

Jonathan Kao

Demetri Terzopoulos, Committee Chair

University of California, Los Angeles

2022

To Ammu and Abbu,

for their constant support

and inspiration to be my best self.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my appreciation to all who made the work of this thesis possible. First, I am grateful for Professor Demetri Terzopoulos' support and guidance in designing this project and providing invaluable feedback. I thank my colleagues in the UCLA Computer Graphics & Vision Laboratory for their help and interactions. I also acknowledge that the work of this thesis is built upon the work of Dr. Masaki Nakada who with Arjun Lakshmipathi created the biomimetic eye model and with Honglin Chen created the LiNets and data collection methods for training the oculomotor control system. My thanks to Arjun and Masaki for their guidance and assistance.

VITA

2017 Full Stack Software Engineering Intern, Project Looma, Village Tech Solutions, Palo Alto, California.

2018 Software Engineering Intern, Device Drivers, Nvidia, Santa Clara, California.

2019 Software Engineering Intern, Device Drivers, Nvidia, Santa Clara, California.

2020 B.S. (Computer Science and Engineering), University of California, Los Angeles.

2020 Software Engineering Intern, Traffic Engineering, Splunk, San Jose, California.

2020–present Teaching Assistant, Computer Science Department, UCLA.
Taught CS 31 (Introduction to Computer Science I) under the direction of Professor David Smallberg (Fall 2020).
Taught CS 32 (Introduction to Computer Science II) under the direction of Professor David Smallberg (Winter 2021, Spring 2021).
Taught CS M51A (Logic Design of Digital Systems) under the direction of Professor Richard Korf (Fall 2021).

2021–present Graduate Researcher, UCLA Computer Graphics & Vision Laboratory, Computer Science Department, University of California, Los Angeles.

CHAPTER 1

Introduction

The human visual system is an astounding computational machine. Photons impact the retina and, through neural processing, enable the visual cortex to perform multiple visual tasks quickly, with high precision, and using very little energy. Artificial neural network (ANN) based computer vision algorithms have come far in recreating the performance of the visual cortex in terms of accuracy, but they employ power-hungry GPUs that lag behind the power and low-latency performance of the human brain.

Spiking neural networks (SNNs) (Jose et al., 2015), comprised of interconnected spiking neurons that, like their biological counterparts, communicate via electrical spikes, are hailed as the "third wave of deep learning." This is because they feature a more biologically-inspired neuronal model that consumes much less power than the standard artificial neurons now commonly in use. Many traditional AI tasks can be achieved with SNNs implemented using the appropriate hardware, which is referred to as "neuromorphic" and currently takes the form of what are known as neuromorphic chips (Bouvier et al., 2019).

This thesis explores the design and training of an SNN in a computer vision task. Our work builds upon that of Lakshmipathi (2018) and Nakada et al. (2019) who developed a biomechanical model of the human eye with a comprehensive set of ocular organs, as well as a neuromuscular oculomotor control system for this realistic simulation model.[1] Of specific interest in this thesis is one of the neural networks within the perception subsystem of the oculomotor controller. The network, described in greater detail in the

[1]This model of the eye was a vast improvement over a simpler eye model created as part of the biomechanical human musculoskeletal model developed by Nakada et al. (2018).

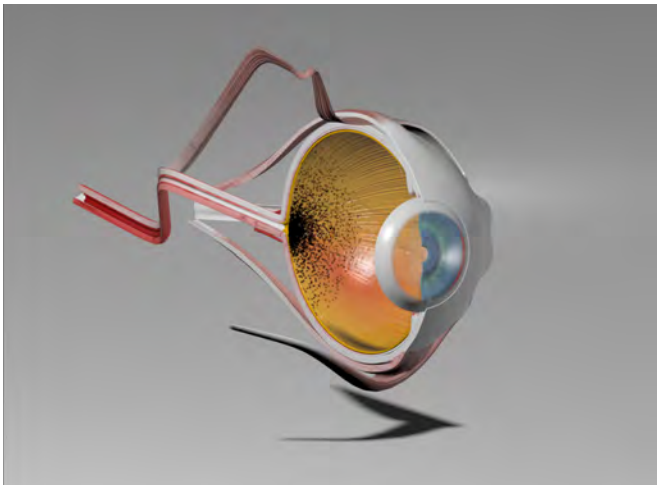


Figure 1.1: (Cross section of a detailed geometric model of the left eye. The black dots on the retina indicate the positions of retinal photoreceptors. Image from (Nakada et al., 2019).

paper by Nakada et al. (2021), was referred to as the foveation LiNet or locally-connected irregular network. While the biomimetic eye model can synthesize realistic eye motion, the "neurons" comprising the artificial neural networks in its oculomotor controller are only high-level abstractions of biological neurons. This thesis aims for additional biological realism through the use of spiking neurons interconnected to form SNNs. To this end, we explore how to encode our floating-point signals into spike trains, which is the job of photoreceptors in the mammalian retina. It remains unclear whether the visual system uses rate or latency encoding (Rullen and Thorpe, 2001), both of which we consider in our work. In addition, we train our SNNs on event-based data, which emulates the ON-OFF bipolar cells of the retina. Our effort is aimed at enhancing the biological realism of the eye model.

1.1 Thesis Contributions

More precisely, the contributions of the thesis are as follows:

1. We devise a novel foveation LiNet, based on SNNs, which we call the SLiNet, to enable the oculomotor control of our biomimetic eye model. Unlike the previous

network for this model (Nakada et al., 2019), ours yields an event-based sensory system that responds only to changes in the light intensities sensed by the eye rather than to the absolute intensity values themselves. This is more biologically accurate and creates a sparser input to the oculomotor controller.

2. To accomplish the above, we design an SNN architecture than can solve a regression task, which is more difficult than the classification tasks to which SNNs have typically been applied.

3. Unlike the typical deployment of previous SNNs, we train our SliNet from scratch. To this end,

    - we consider rate and latency encoding, the two most commonly used encoding methods, and find the best encoding parameters for each method, and
    - we use a surrogate gradient to solve the "dead neuron problem" and enable the use of standard deep learning optimization techniques.

4. The work of Nakada et al. (2019) and the MS thesis of Lakshmipathi (2018) developed the biomimetic eye model and demonstrated its biological accuracy by testing it on different types of eye movements (saccade, fixation, and smooth pursuit). We use the same experimental regimen to test our SLiNet's performance on both conventional and event-based data.

1.2 Thesis Overview

The remainder of this thesis is organized as follows:

Chapter 2 surveys the relevant literature related to the use of SNNs in computer vision.

Chapter 3 prescribes the visual tracking task of interest and presents the details of the biomimetic eye model with which we work.

Chapter 4 compares artificial neurons to spiking neurons and explains how SNNs

operate, which includes exploring how to encode our inputs and interpret the output spike trains.

Chapter 5 describes the architecture of our SNN as well as the associated training techniques.

Chapter 6 reports our results and compares our SNN performance to that of the previous LiNet architecture.

Chapter 7 presents our conclusions and suggests promising avenues for future work.

CHAPTER 2

Related Work

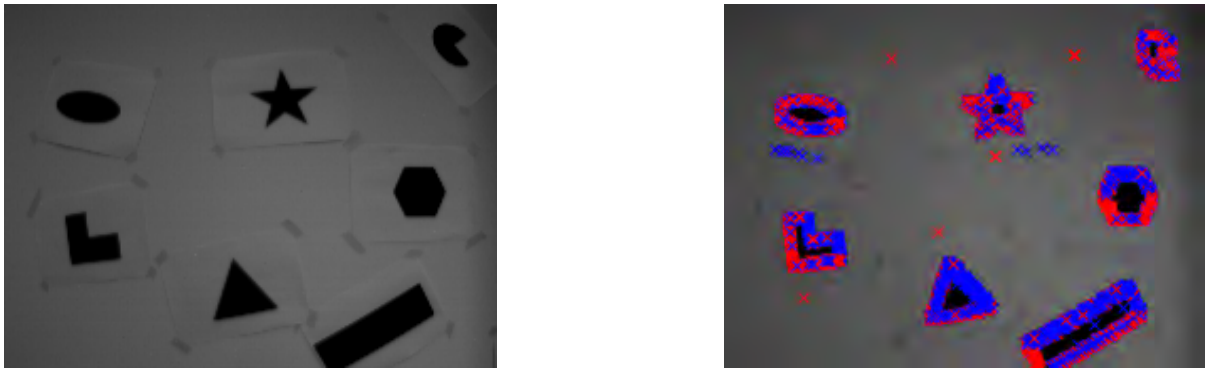
In this chapter, we review relevant prior research related to SNNs.

2.1 Neuromorphic Hardware

Currently, graphics processing units (GPUs), the workhorses of deep learning based artificial intelligence using ANNs, are optimized for highly parallelized multiply-and-accumulate (MAC) operations on floating-point numbers and consume large amounts of power. By contrast, neuromorphic chips take advantage of the fact that spiking activation functions only output 1's and 0's and hence operate with low power consumption. They are also optimized for the asynchronous nature of spikes and can run new types of learning algorithms.

Several major corporations are investing in the design and manufacture of neuromorphic chips. Intel's Loihi (Davies et al., 2018) is on its third generation, and IBM, Samsung, and other large companies are betting on the future of this sector. However, because this hardware is still difficult to acquire, we run our SNNs on a GPU.

In addition to neuromorphic chips, neuromorphic sensors are also under development. Vision has been the sensory modality to receive the most attention, and dynamic vision sensor (DVS) cameras, or "silicon retinas," are slowly entering more and more systems. Instead of capturing data one frame at a time, these cameras simply record if pixels in their sensory arrays have gotten brighter or darker. If such a change occurs at any pixel, the camera outputs a timestamped "event." The DAVIS sensor (Brandli et al., 2014) is one example of a DVS camera. New spiking datasets are being created with DVS sensors,



(a) (b)

Figure 2.1: This dataset records a DVS camera moving around a collection of shapes. The camera has moved slightly between the frames in (a) and (b). The resulting changes in pixel intensities are shown in (b), where pixels that have brightened are marked in red while those that have darkened are marked in blue. Images from (Mueggler et al., 2017).

as shown in Figure 2.1.

2.2 Converting Artificial Neural Networks into SNNs

Many researchers want the benefits of SNNs, but they also prefer to avoid training them. This has prompted research into converting trained artificial neural networks (ANNs) into SNNs. The main advantage of this approach is that one need not work around the non-differentiability of the spiking activation function and can effectively train a model with standard deep learning techniques.

Diehl et al. (2015) are credited with developing early conversion techniques for fully connected networks. Now, almost any existing neural network layer can be converted into a spiking equivalent, including convolution and softmax layers (Rueckauer et al., 2017). These converted models may have slightly higher error rates, but they can offer about a 2x reduction in the number of operations when compared to the original ANNs.

In this thesis, however, we aim to train SNNs directly rather than convert them from trained ANNs. We take this approach not just because converted SNNs typically require neurons to have high spiking frequencies, which results in the consumption of more power, thereby undermining the benefits of using SNNs in the first place. More importantly, we

aim to explore event-based data in a biologically plausible setting. Converted models are typically trained on data with events calculated after the fact and they perform poorly when fed more realistic spiking data from a DVS sensor.

We also open the door to the use of more biologically-inspired, unsupervised learning techniques to train our model in the future. In particular, spike timing dependent plasticity (STDP), or Hebbian learning, has shown great promise (Kheradpisheh et al., 2018). Winner-take-all (WTA) circuits that make use of inhibition are also of much interest.

2.3 SNNs and Computer Vision

Traditional computer vision tasks are being addressed with SNNs. MNIST handwritten digit classification remains a popular benchmark (Diehl and Cook, 2015), but SNNs also perform well on more complex datasets such as ImageNet. Sengupta et al. (2019) develop an SNN based on VGGNet and achieve a top-5 error rate of 30.04, whereas the state-of-the-art ANN achieves a top-5 error rate of 29.48. Other complex models such as ResNet have been trained to work directly with spiking input from a DVS camera (Maqueda et al., 2018).

Most computer vision research with SNNs to date has chosen to work with classification problems. It is often stated in the literature that SNNs are easier to apply to classification problems than to regression problems. This is due to the fact that there is consensus on how to interpret spike trains so as to classify an input, but there are many choices in how to interpret spike trains to represent continuous quantities. The only published work on regressions with SNNs is by Gehrig et al. (2020) and Kim et al. (2020). However, their SNN models were converted from a trained ANN. We train our SNN from scratch and offer an alternative approach to creating an output layer for a regression problem.

Our eye model has a biomimetic, nonuniform photoreceptor distribution, whereas the pixels in images from datasets such as MNIST and CIFAR-10 are conventionally arranged in structured arrays. Permutation invariant MNIST, or PI-MNIST (Le et al., 2015), is an

interesting dataset that alters the arrangement of pixels so that standard convolutional layers cannot be used to classify digits and serves as the only benchmark that remotely resembles the unstructured input of the retinal model of interest in this thesis.

CHAPTER 3

The Task

Our goal is to create an SNN that, when properly trained, enables the biomimetic eye model of Nakada et al. (2019) to track a 3D object through space. The visual target of choice is a white ball that moves against a background whose intensity varies in different shades of grey due to changing illumination. The eye model's oculomotor controller has four main parts, one of which we implement as an SNN. We test our implementation by checking that our modified controller performs certain movements with biological accuracy and that it can successfully track the target. In this section, we provide more details about the eye model and its oculomotor controller and focus on the retina and the foveation deep neural network (DNN) that we replace with an SNN.

3.1 Retina

Like a biological retina, our virtual retina is situated at the back of the eye and has photoreceptors that sense light from the scene. Each photoreceptor collects a red, green, and blue color value. The N photoreceptors are nonuniformly distributed according to a noisy log-polar distribution

$$d_k = e^{\rho_j} \begin{bmatrix} \cos(\alpha_i) \\ \sin(\alpha_i) \end{bmatrix} + \begin{bmatrix} \mathcal{N}(\mu, \sigma^2) \\ \mathcal{N}(\mu, \sigma^2) \end{bmatrix}, \text{ for } 1 \leq k \leq \mathrm{N}, \tag{3.1}$$

where $\mathcal{N}(\mu, \sigma^2)$ denotes IID-sampled Gaussian noise of mean μ and variance σ^2. This distribution places most of the photoreceptors near the center of the retina, forming a foveal region that supports high acuity central vision, with progressively diminishing

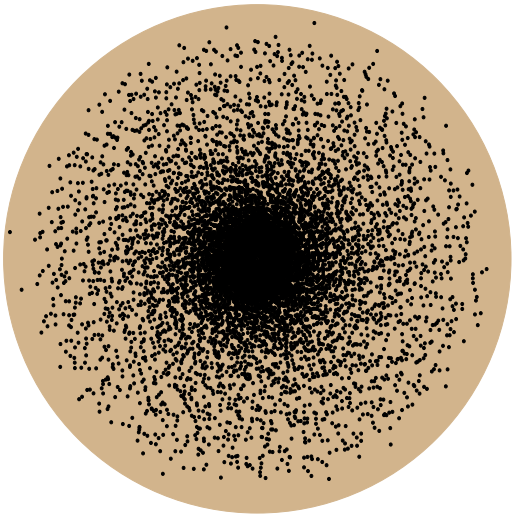


Figure 3.1: The photoreceptors on the retina are arranged according to a 2D noisy log-polar distribution, which is mapped to the hemispherical fundus of the eyeball.

visual resolution towards the retinal periphery.

A visualization of the photoreceptor distribution on the retina is shown in Figure 3.1. Here, we have chosen $N = 14{,}400$ photoreceptors in (3.1), incrementing ρ_j and α_i in steps of 1 such that $0 \leq \rho_j \leq 40$ and $0 \leq \alpha_i \leq 360$. The additive Gaussian noise distribution has mean $\mu = 0$ and variance $\sigma^2 = 0.0025$. Using fewer photoreceptors speeds up simulation and training, but the number of photoreceptors can be scaled up to match human retinas (which have about 6 million cone photoreceptors supporting normal color vision and about 120 million rod photoreceptors supporting monochrome low-light vision (Purves et al., 2001)).

To compute the amount of light registered by each photoreceptor, in accordance with the ray tracing procedure that is well-known in computer graphics (Shirley and Morley, 2003), rays are cast from the positions of photoreceptors on the retinal surface, refracted through the deformable lens of the eye, through the pupil, again diffracted through the cornea, and out into the 3D environment to recursively intersect with environmental

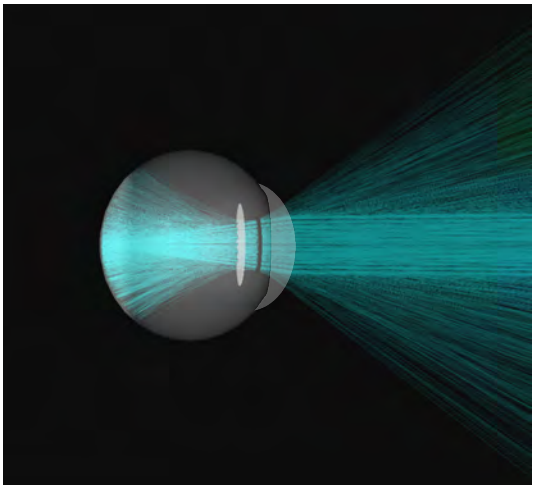


Figure 3.2: Rays cast from the positions of the photoreceptors through the finite-aperture pupil and out into the scene to compute the irradiance responses of the photoreceptors according to the ray tracing process. Image from (Nakada et al., 2019).

objects in the scene and sample the lightsources (Figure 3.2). The computed color values returned from the recursion determine the irradiance at each photoreceptor, thus resulting in 14,400 RGB values. We stack the RGB values of each photoreceptor into a $3N = 43{,}200$-dimensional vector, which Nakada et al. (2018) refer to as the optic nerve vector (ONV).

3.2 LiNets

The next step is one of visual processing analogous to that taking place in the visual cortex of the brain. Convolutional neural networks (CNNs), which abstractly model the connectivity of neurons in the visual cortex, have enabled much progress in computer vision. In CNNs, each neuron is connected only to its neighboring neurons in the previous layer, thereby forming what are known as "receptive fields". The stylized, highly regular receptive fields of conventional CNNs exploit the fact that ordinary images are structured as rectangular arrays of pixels.

By contrast, the biomimetic photoreceptor distribution on the retina in our eye model is an irregular, foveated distribution, and the ONV exiting the retina is simply a vector of photoreceptor responses rather than a 2D pixel-array image. Therefore, the ONV

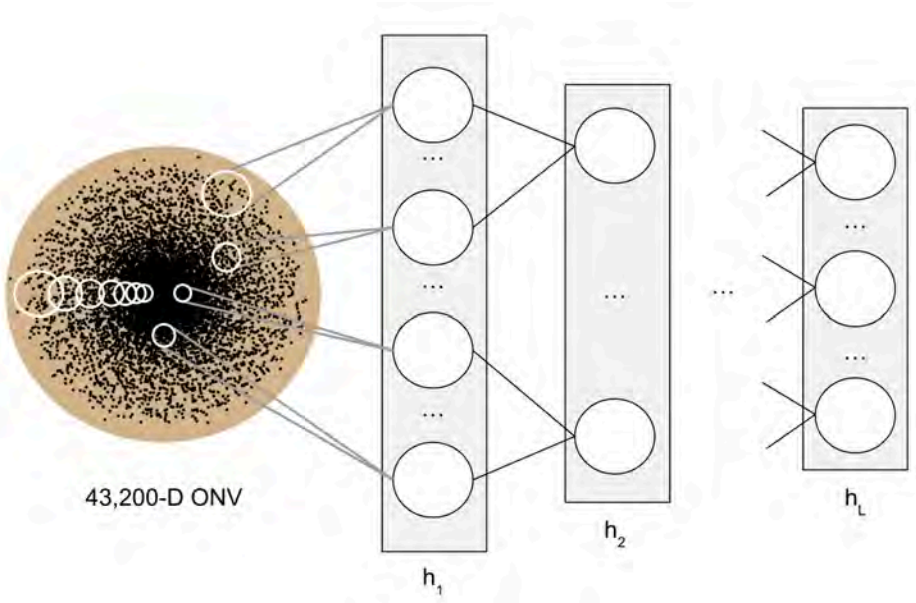


Figure 3.3: The LiNet architecture. We have a 14,400 photoreceptor retina that creates the input ONV. Neurons at the first layer combine input from a handful of photoreceptors. Because photoreceptors are more densely packed towards the center of the retina, receptive fields get larger towards the periphery. Neurons in consecutive layers combine the fields of view from previous layers to slowly get closer to seeing the full scene.

is incompatible with CNNs. Consequently, Nakada et al. (2019) generalized CNNs by introducing locally-connected irregular networks or "LiNets" (Nakada et al., 2021). The LiNet architecture is illustrated in Figure 3.3. Neurons have associated positions within the visual field. According to these positions, each neuron is connected only to the n nearest neurons in the previous layer, thus forming overlapping circular receptive fields at the retinal level, which are shown as white circles in the figure. The number of neurons in successive layers is scaled down by a factor f.

Like CNNs, LiNets consume far less memory than comparably-sized fully-connected networks, thus accommodating retinas with large numbers of photoreceptors. However, unlike CNNs, the receptive fields of neurons within a given layer do not share weights (i.e., they are not convolutional), so the memory requirements of LiNets are generally greater than those of CNNs.

3.3 Oculomotor Control

In addition to the retina, lens, pupil, and cornea, the biomimetic eye model includes the 6 extraocular (EO) contractile muscles. The cornea and deformable lens focus visual targets onto the retina while the EO muscles drive the eye movements necessary to foveate and track visual targets in motion. Each muscle requires a time-varying motor activation signal that stimulates it to contract. An oculomotor controller is responsible for producing the muscle activation signals that drive the eye movements needed to accomplish the visual task of interest.

Additional details about the eye model and its oculomotor control system are provided in Appendix A. The latter comprises a sensory subsystem and a motor subsystem, as shown in Figure A.1. The ONV is input to the foveation DNN, which is implemented as a LiNet. This foveation LiNet outputs 2 values, $\Delta\theta$ and $\Delta\phi$, that represent a desired change in the horizontal and vertical gaze angles relative to the eye's current gaze direction. Figure 3.4 presents an example of how we encode the motion of a white ball visual target with these two gaze angles. The neuromuscular DNN is then fed the outputs of the foveation LiNet and outputs an activation signal for each of the 6 EO muscles to induce the required eye movement.

There are two viable methods for synthesizing eye movements—neuromuscular control as employed by Nakada et al. (2019) and inverse dynamic control. Because this thesis deals specifically with the foveation LiNet, we synthesize eye movements using inverse dynamics control. This starts with computing the angular position in which the eye needs to be in order to keep up with the moving visual target. Given the current angular position and desired angular position, we can compute the angular acceleration needed to complete the movement. From the angular acceleration we can compute the torque that the muscles need to produce and, finally, the muscle activations necessary to produce the desired torque. Although inverse dynamics control is less realistic than neuromuscular control, our goal is simply to prove that our SNN-based foveation network—dubbed the SLiNet—matches the performance of the existing LiNet foveation network.

(a) (b)

Figure 3.4: The ball moves from one point (a) to another (b). The red line reveals the current gaze direction, which shifts to the left and slightly downward between the two frames. This horizontal and vertical movement is encoded in the angles θ and ϕ, respectively.

To this end, we will test our SLiNet in accordance with the experimental regimen of Nakada et al. (2019). This involves moving a visual target in three different ways and noting if the eye can successfully track it. The eye movements are defined as follows:

1. *Fixation*: The eye foveates the target while it remains fixed in space. The eye appears still and stable but may make small oscillatory movements. We test that these small oscillations exist as they enhance the biological plausibility of the model.

2. *Saccade*: A quick eye movement that brings the visual target from peripheral vision to the central, foveal region of the retina. We test the extent to which our trained foveation SLiNet yields angular displacement, velocity, and acceleration similar to the previous LiNet model.

3. *Smooth pursuit*: Once the visual target is fixated upon, the eye can pursue the target as it moves freely in space, including approaching or receding from the eye. The trajectory of the eye movement should include realistic oscillations. We track the orientation of the eye through time to make sure that it successfully follows the target. Good performance here is an indication that the eye can track the target through many different trajectories.

CHAPTER 4

Spiking Neurons

In this chapter, we introduce the basics of spiking neural networks. Additionally, we detail how to encode inputs to spiking neurons and interpret their outputs to perform a regression.

We first define the popular ANN currently in use. The corresponding neuron model is summarized in Figure 4.1. In an ANN with L layers, W^l, for $l \in 2, \ldots, L$, is the weight matrix connecting layer $l-1$ to layer l. We multiply the output of the previous layer, x^{l-1}, with W^l, add biases b^l:

$$a^l = W^l x^{l-1} + b^l, \tag{4.1}$$

and finally apply a rectified linear unit (ReLU) activation function:

$$x^l = \max(0, a^l). \tag{4.2}$$

Before moving on to a spiking neuron, which is illustrated in Figure 4.2, we define some terms. A synapse refers to the connection between two neurons which has an associated weight w that is tuned during training. Looking at a specific neuron, all neurons in the previous layer that are connected to it are referred to as presynaptic neurons and all neurons connected to it in the next layer are postsynaptic neurons. The inputs are time varying and in the form of spike trains, which are sequences of 1's and 0's. All spike trains in the network are the same length, which is treated as a hyperparameter.

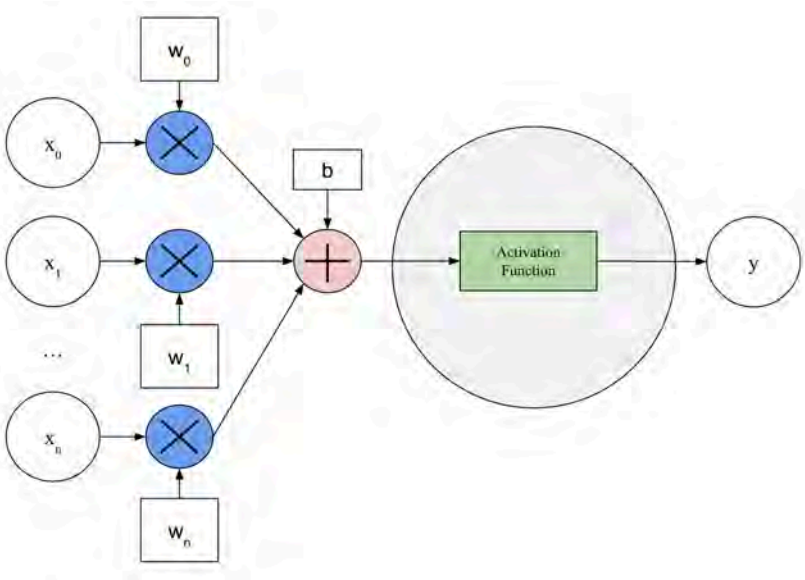


Figure 4.1: Internals of a conventional artificial neuron. The inputs, weights, and output are usually floating point numbers, whose multiplication requires multiply and accumulate (MAC) instructions, which typically consume considerable power.

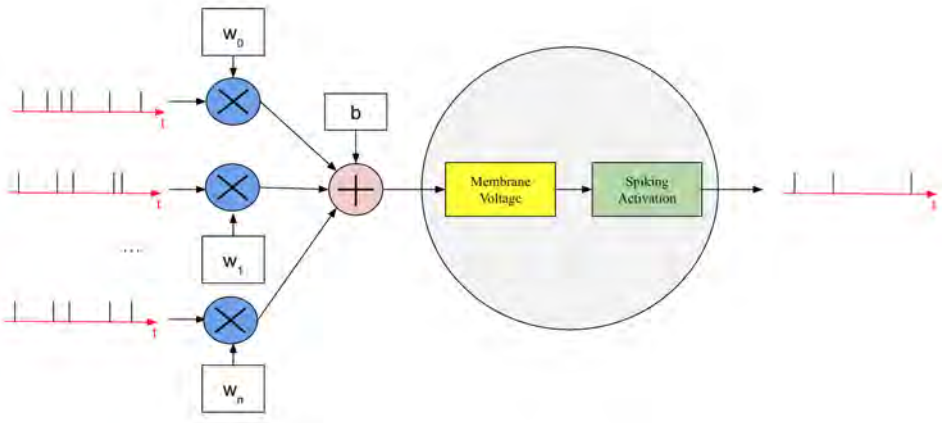


Figure 4.2: Internals of a spiking neuron. Each weight can only be multiplied by a 1 or a 0, eliminating the need for expensive MAC instructions.

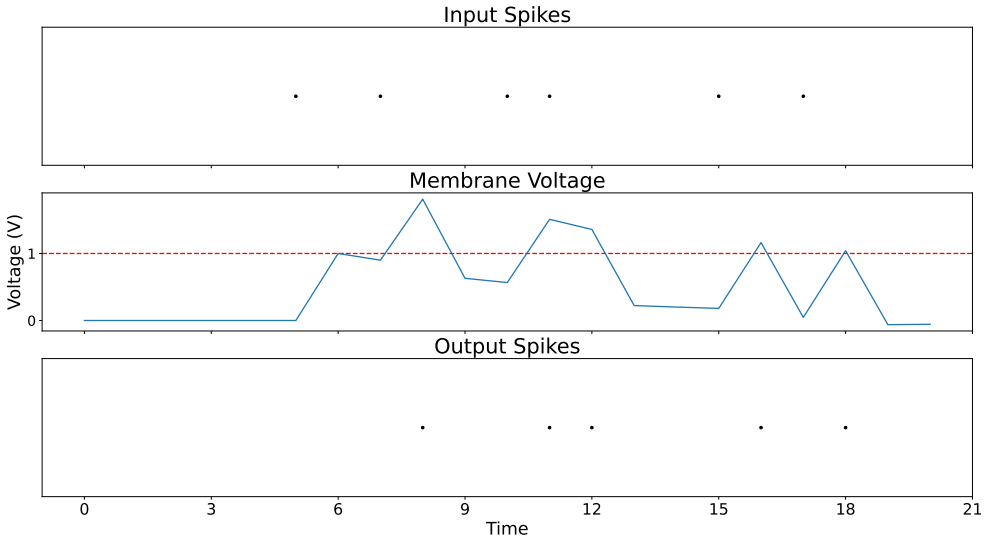


Figure 4.3: Demonstration of a LIF neuron. Input spikes to the neuron are at the top, the neuron's membrane voltage is in the middle, and output spikes are at the bottom. The threshold voltage is denoted by the dashed red line. We verify that the membrane voltage increases when there is an input spike and that there is an output spike when the membrane voltage crosses the threshold. A reset occurs after each output spike, represented by the steep decrease in the membrane voltage. There is also a slight leak of membrane voltage in the absence of any input spikes (such as around timestep 6).

Each spiking neuron maintains a state variable known as the membrane voltage

$$U(t) = \underbrace{\beta U(t-1)}_{\text{decay}} + \underbrace{WX(t)}_{\text{input}} - \underbrace{S(t-1)U_T}_{\text{reset}}, \tag{4.3}$$

where matrix W stores the weights and tensor X stores the presynaptic inputs, and where the spiking activation function is $S(t)$.

$$S(t) = \begin{cases} 1 & \text{if } U(t) > U_T \\ 0 & \text{otherwise} \end{cases} = \mathcal{H}(U(t) - U_T) \tag{4.4}$$

$U(t)$ used to calculate the membrane voltage of a neuron at timestep $t+1$. If a presynaptic neuron spikes, we add the corresponding synapse weight to this membrane voltage. If no spikes are input to the neuron, the membrane voltage decays exponentially. This decay

is controlled with the hyperparameter β. If U exceeds a certain threshold U_T, then the neuron outputs a spike and resets U to zero. This spiking behavior is captured by the Heaviside step function, represented by $\mathcal{H}$.

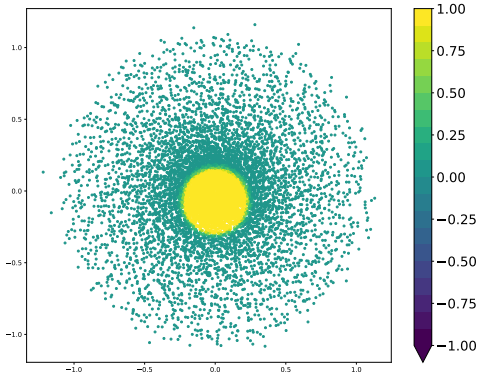
The membrane voltage equation is used by snnTorch (Eshraghian et al., 2021), a Python package built on top of PyTorch, which we employ in our work. This neuron model is known as the leaky integrate-and-fire (LIF) neuron, so named because it "leaks" voltage in the absence of an input. Figure 4.3 demonstrates how membrane voltage is calculated with an example LIF neuron.

To summarize, an SNN differs from the conventional ANN in two fundamental ways. First, we replace the activation function with one that only outputs 1's and 0's. Second, we have inputs that vary over time. More detailed derivations of SNN theory can be found in Appendix B.

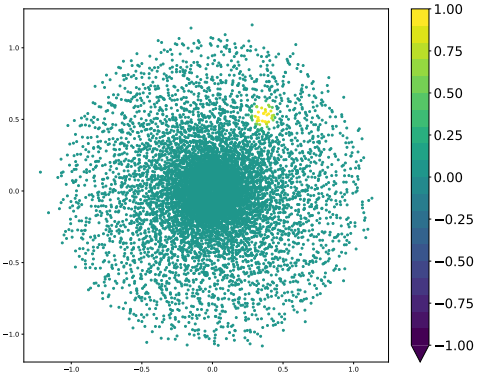
4.1 Encoding Input Signals

As stated previously, our ONV is a vector of dimension 43,200. SNNs, however, expect inputs that vary over time. In addition, the floating point numbers that represent the RGB light intensity at any retinal photoreceptor need to be meaningfully converted into "spikes;" i.e., a series of 1's and 0's. This is known as converting data from the "frame" domain to the "spiking" domain, and we explore two main conversion schemes.

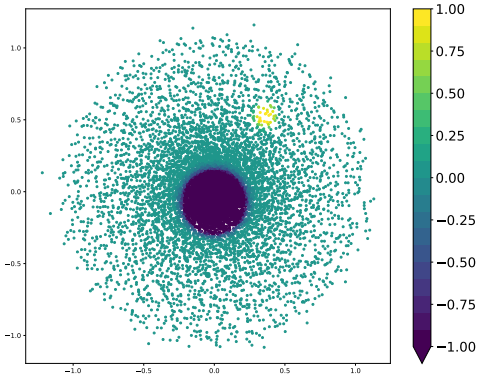
In addition to generating spiking inputs, we also introduce what we refer to as the Delta-ONV, or D-ONV for short; instead of having the eye see light intensities at the current timestep, it sees the *difference* between the current and previous ONV values, as shown in Figure 4.4. In other words, the eye detects only the changes in the scene that manifest in intensity changes at the retinal photoreceptors. These changes are also referred to as "event-based" data. Note that the D-ONV exhibits positive values at photoreceptors that register brighter and negative values if they register darker. This results in sparse input data as the eye need not repeatedly re-process what it has already observed. The D-ONV is more biologically accurate, since ganglion cells in the retina



(a) Previous ONV



(b) Current ONV



(c) D-ONV

Figure 4.4: We map ONV values collected from a scene by the photoreceptor distribution. Note that ONV values are in the range $[0, 1]$ while D-ONV values are in the range $[-1, 1]$. The target has moved from one position in (a) to another in (b). (c) shows the corresponding D-ONV. The values darken where the target was in (a) and get brighter at the target's location in (b).

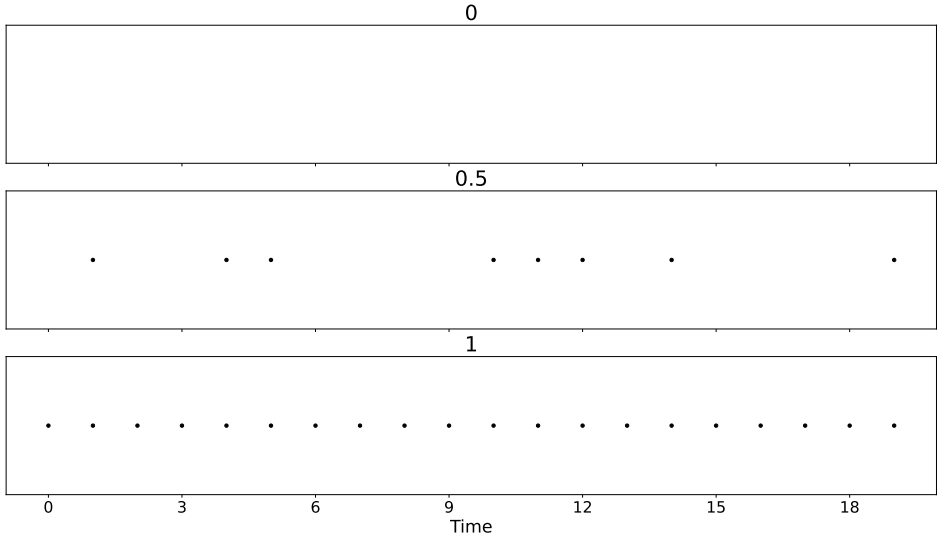


Figure 4.5: Demonstration of rate encoding with Bernoulli trials. From top are encoded data values of 0.0, 0.5, and 1.0.

emit spikes only when there is an intensity change in the field of view (Tayarani-Najaran and Schmuker, 2021).

Next we review the two most popular encoding methods, rate and latency encoding. It is still unknown exactly how neurons encode various stimuli into spikes, but it is believed that each method is used in different parts of the brain, including in the visual cortex. Wang et al. (2016) find that retinal ganglion cells use different encoding methods for different tasks. We empirically found that rate encoding performed better than latency encoding on our object tracking task.

4.1.1 Rate Encoding

Rate encoding attempts to encode a neuron's firing frequency. Each input value to the encoder lies in the range between 0 and 1, representing the probability that the neuron will spike at a given timestep. Then, at each timestep, we run a Bernoulli trial with the given probability to determine if the neuron will spike. For example, if our photoreceptor intensity value was 0, the neuron would never spike. On the other hand, a value of 1 would create spikes at each timestep. A value of 0.5 would result in approximately half of

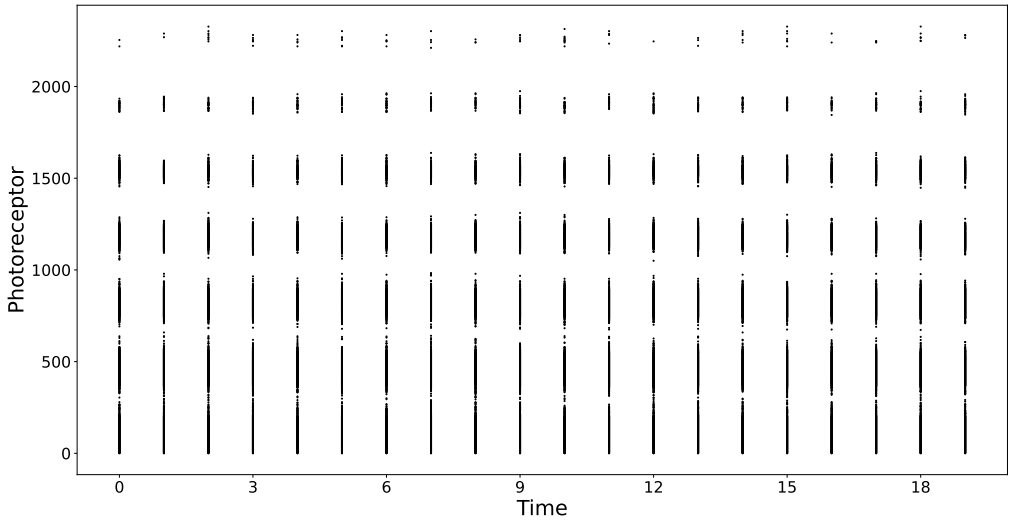


Figure 4.6: Rate encoding of a subset of photoreceptors from the ONV in Figure 4.4a. Each horizontal line represents the spiking behavior of one photoreceptor. Only the photoreceptors with nonzero inputs are spiking, and we can see different spike rates corresponding to different light intensities.

the timesteps containing a spike. We show these examples over 20 timesteps in Figure 4.5.

Each of our RGB color channels are already in the range of 0 to 1, so they can directly be rate encoded. In Figure 4.6 we present a subset of the input spikes that result from rate encoding the ONV from Figure 4.4a. We can see a few different firing rates in the figure, with excited neurons firing at every timestep and other neurons not firing at all.

With the D-ONV, however, we have values in the range of -1 to 1. This is a problem because we cannot have a negative probability assigned to whether or not an input neuron will spike. The solution is to take the absolute value of the probability, and if a spike is generated, it will carry a value of -1 instead of 1. This means that neurons in the next layer will decrease their internal voltages if they receive a spike with value -1. Figure 4.7 shows what happens when a neuron receives negative spikes as input.

The inputs to the rate encoder can also be scaled up before being turned into spikes. This is referred to as the gain, which we treat as a hyperparameter. For example, given gain g and input x, our new input becomes $g \times x$. Values above 1 are clipped to 1.

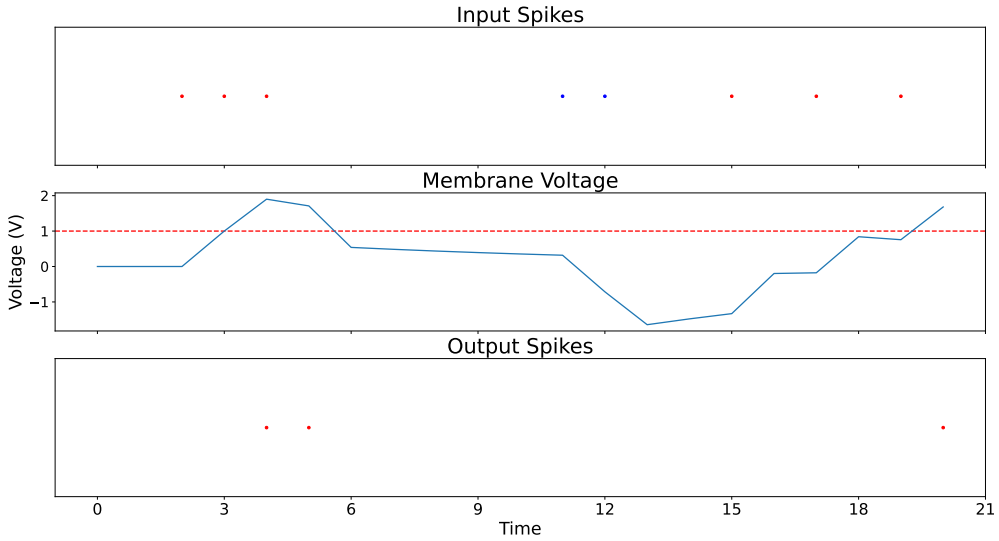


Figure 4.7: A LIF neuron that receives both positive (red) and negative (blue) spikes as input. The membrane voltage increases with positive input spikes and crosses the threshold voltage to output spikes. Around timestep 5 the neuron resets, but there is also another input spike. This spike adds to the membrane voltage, causing it to stay above the threshold after the reset. When the negative spikes arrive as input, the membrane voltage decreases.

Larger gain values seemed to create a smoother decrease in loss and better training performance. This makes sense intuitively because more spikes are created in the input layer, giving downstream neurons more opportunities to fire. However, larger gain values also lead to higher validation loss and result in models that cannot track the target during inference. We limit the amount of gain to 2.0, meaning that photoreceptors with values 0.5 and above spike at every timestep.

4.1.2 Latency Encoding

Latency encoding focuses on the timing of spikes rather than the spiking frequency. Each neuron is allowed to fire once in the simulated time interval; neurons with higher probabilities of firing emit their spike earlier than neurons with lower probabilities. This encoding method results in sparser inputs to the SNN when compared to rate encoding and, consequently, also makes it difficult for the model to converge.

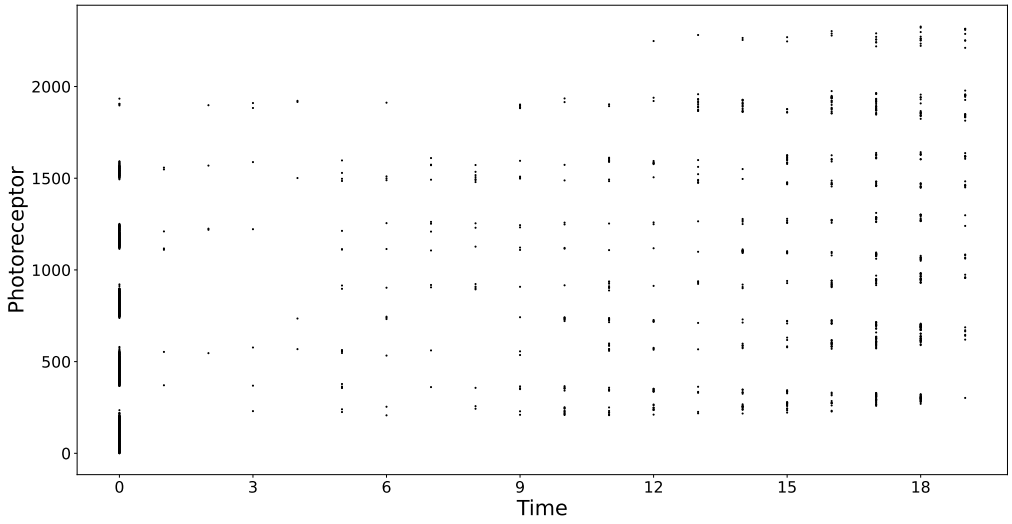


Figure 4.8: Latency encoding of a subset of photoreceptors from the ONV in Figure 4.4a. Each horizontal line represents the spiking behavior of one photoreceptor, which spikes once in the 20 timestep interval. Excited photoreceptors spike earlier than less excited ones, which spike later.

Figure 4.8 contains spikes that result from latency encoding the ONV from Figure 4.4a. In the figure, we see that very excited neurons fire at the first timestep and partially excited neurons fire sometime afterwards. The spikes at the last timestep represent inputs of 0.0.

4.2 Outputs

A problem associated with using SNNs is related to interpreting the output spike trains. For classification problems, each output neuron is associated with one possible class. The output spike trains are then integrated over a number of timesteps and the output label is that of the neuron with the most spikes, as illustrated in Figure 4.9.

Our application in this thesis, however, takes the form of a regression problem that involves outputting predictions for modifying two gaze angles. Thus far, SNN research has focused on classification as there is no standard way to interpret spikes into floating point values.

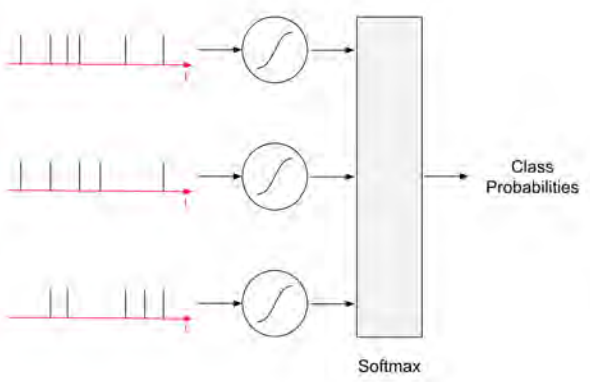


Figure 4.9: An example on how spike trains can be interpreted for the purposes of a classification task. Here, each spike train is integrated to count the spikes during a time interval. These counts are then passed to a softmax layer. Each neuron is assigned to a possible class, and the predicted label is that of the neuron with the most spikes.

To solve this regression problem, we utilize a linear layer to transform outputs from our SNN into the desired angular modifications $\Delta\theta$ and $\Delta\phi$. One option would be to consolidate the output spike train somehow, such as with a sum or average, and then normalize the values. A technique that has not been explored much, but is suggested in the snnTorch package, is to utilize the membrane voltages of the last layer. We pass each neuron's membrane voltage at the last timestep through a linear transformation that outputs our two angular changes. This technique helps with backpropagation learning, as the network can learn what membrane voltages it should target to have at the end of the computation. Note that the network outputs values at each timestep, but our predicted values are the model's outputs at the final timestep.

CHAPTER 5

The SLiNet Model

In this chapter, we discuss how the architecture of our foveation DNN is based on that of the LiNet, which was described in Section 3.2.

5.1 Architecture

We base our SNN architecture on the LiNet that was designed for this task by (Nakada et al., 2021). The existing LiNet DNN has 5 locally-connected layers plus one final fully-connected layer, as illustrated in Figure A.4. Each layer has 1/5 the number of neurons of the previous layer.

To build our spiking LiNet, or SLiNet, we start with a 4 layer foveation LiNet and replace the ReLU units with spiking neurons. We retain the fully-connected output layer to transform the membrane voltages into the two gaze angles. Our SLiNet model is summarized in Figure 5.1. We compare code samples implementing an ANN and an SNN in Figure 5.2.

5.2 Training

Here we go over the various decisions made while training our SLiNets.

Our training data set consists of 22.5k data points. We use 20k to create a training set and set aside 2.5k for validation. We do not create a testing set as we evaluate our work through the simulation of the eye model. The data points are collected from the eye model itself. It is kept in a fixed position, looking forward. The ball visual target is

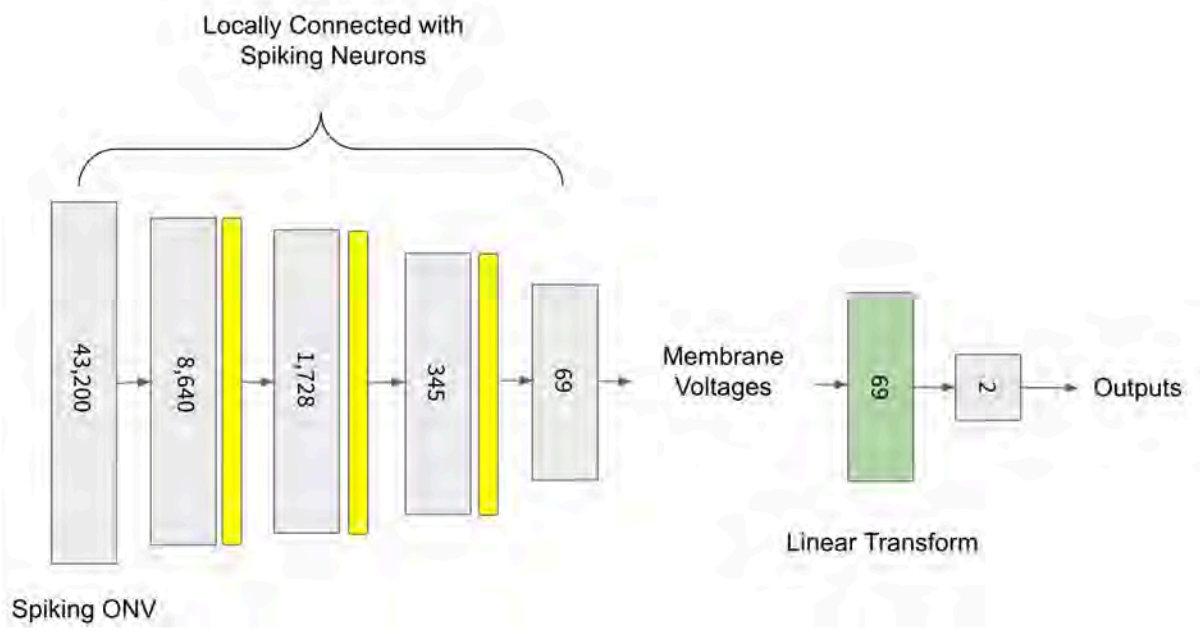


Figure 5.1: Our spiking foveation LiNet architecture, the SLiNet. The first four layers reflect that of the LiNet, but employ spiking neurons (yellow) rather than the conventional neurons with ReLU activation functions. The fourth layer passes the membrane voltages through a linear transform, which produces an output of dimension two, representing the changes in gaze angles $\Delta\theta$ and $\Delta\phi$.

moved to random points in the field of view, and the corresponding ONVs are collected. The labels are the angular displacement between the eye's current gaze direction and the direction of the ball in the field of view. To create our D-ONV data set, we simply subtract each ONV from the previous one.

For the LiNet, we use a factor f of 5, meaning each subsequent layer has one-fifth the number of neurons as the previous layer. A number was not reported for k, the number of inputs combined at each neuron. We conducted a hyperparameter sweep and use a value of 25. We re-trained the LiNet since we did not have access to the previously trained model. Training was done with a batch size of 16 and a learning rate of 0.001. No regularization was added to the model, and the weights were initialized with He initialization. The same hyperparameter values are used when training our SLiNet.

We train our models for around 100 epochs, which is about when the loss values start to plateau. We plot the training and validation losses for both a 5-layer LiNet

```python
def forward(self, x):
  x = self.fc1(x)
  x = torch.relu(x)

  x = self.fc2(x)
  x = torch.relu(x)

  # Output layer
  out = self.fc3(x)
  return out
```

(a)

```python
def forward(self, xTensor):
  # Init LIF Layers to replace Relu
  mem0 = self.lif1.init_leaky()
  mem1 = self.lif2.init_leaky()

  out = None

  for t in range(numSteps):
    x = xTensor[t]

    x = self.fc1(x)
    x, mem1 = self.lif1(x, mem1)

    x = self.fc2(x)
    x, mem2 = self.lif2(x, mem2)

    # Output layer
    out = self.fc3(mem2)
  return out
```

(b)

Figure 5.2: Code comparison between a fully-connected ANN (a) and SNN (b). Differences in (b) include the addition of spiking neurons, time-varying input, and the use of a linear transform that transforms membrane voltages from the last spiking layer. Note that "lif" is a layer of spiking neurons and that "fc" is a fully connected layer. A LIF neuron outputs a tuple with two values: a spike (or a lack thereof) and the current membrane voltage.

and our 4-layer SLiNet. Figure 5.3 shows plots for each model trained with the ONV and Figure 5.4 shows plots for each model trained with the D-ONV. In both plots we can see that the LiNet achieves loss values one order of magnitude smaller than the spiking models. However, the spiking models still converge to low-loss values. The LiNet validation loss is much higher when using the D-ONV, which may indicate that it will have trouble generalizing to our test scenarios.

We treat β as a hyperparameter, although it can also be trained in different ways. We choose not to tune it on a per-layer or individual neuron basis with backpropagation. However, we do treat the threshold voltage for each neuron as a trainable parameter. By default, all neuron thresholds are set to 1.0, but we find that randomly initializing these thresholds to values in the range $[0, 1]$ yields the best results.

When a neuron outputs a spike, we can either reset the membrane voltage to zero

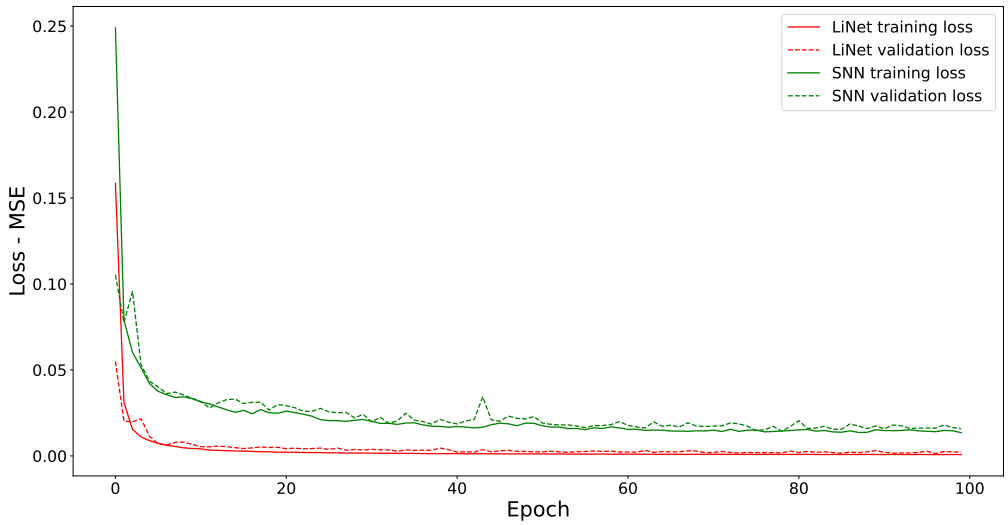


Figure 5.3: The LiNet converges very quickly and achieves very low loss values. There is no indication of overfitting from either model as the validation losses continue to decrease with the training losses.

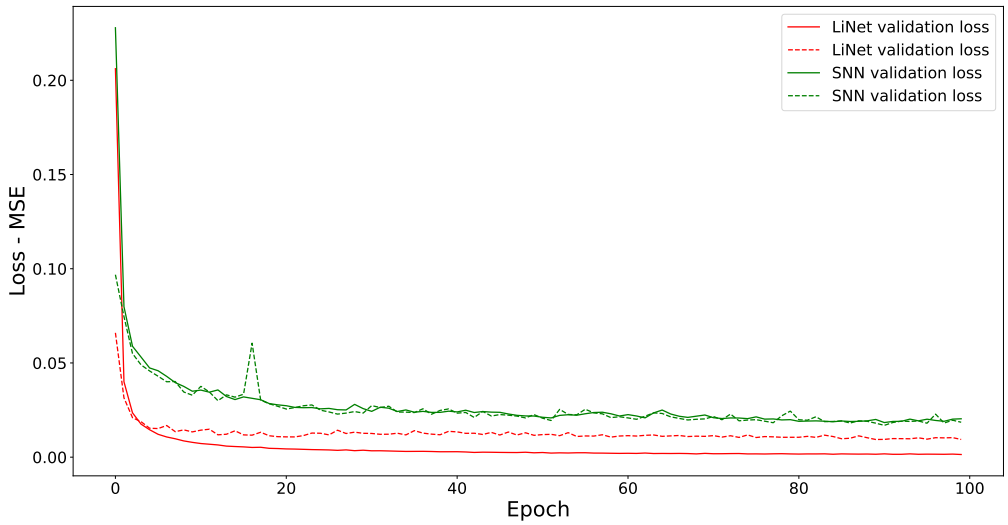


Figure 5.4: When using the D-ONV, the LiNet again converges quickly, but has high validation loss, whereas our SLiNet still converges to reasonably low loss values.

or subtract the threshold voltage from the current membrane voltage. We refer to these options as "reset" and "subtract," respectively. With the subtract method, the neuron will have a non-zero membrane voltage if it had accumulated a large amount of voltage before the spike. On the other hand, the resetting method results in sparser spiking and potential energy savings, but it presents a bigger challenge to learning as it is more lossy. We use the subtraction method in our neurons.

5.2.1 Number of Timesteps

The length of the spiking input, otherwise referred to as the number of timesteps, is a hyperparameter of our SLiNet. This hyperparameter determines how many timesteps each neuron is afforded to accumulate voltage and emit spikes. More timesteps allow more downstream neurons to fire, which may help computation, but will also consume more energy to run. Conversely, models using a lower number of timesteps may train quickly, but they will have difficulty converging to a low enough loss value. The typical number of timesteps ranges from the hundreds to thousands. After conducting a hyperparameter sweep, we empirically found that 20 timesteps was enough for our model to converge. Higher numbers of steps were more difficult to train and did not seem to affect performance.

5.2.2 Surrogate Gradients

On the backpropagation pass, we encounter the Heaviside step function through our spiking activation function. Its local derivative is 0 everywhere except for the time of the spike, where it is infinite. This is the main barrier to deep learning with spiking neural networks, as our gradient will either become 0 or infinity after reaching this function on the backward pass. snnTorch handles this by passing through the gradient when there is a spike and 0 when there is not. This enables some learning, but is not good enough for our task. We experimented with surrogate gradients, which are functions that approximate the Heaviside step function but are differentiable everywhere (Neftci et al., 2019). The spiking activation is still used in the forward pass, but the surrogate gradient is used on

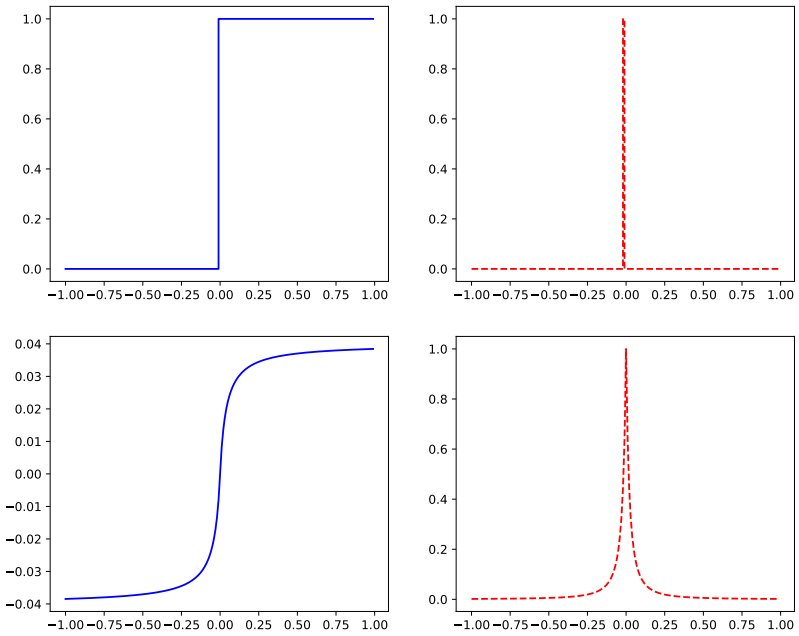


Figure 5.5: We compare the Heaviside step and fast sigmoid functions (blue) and their derivatives (red). The derivative of the Heaviside step function is 0 everywhere except at the location of the threshold, where it is infinity. This is problematic during backpropagation, where the derivative will either be set to zero or explode. By contrast, the derivative of the fast sigmoid function is smooth and allows gradients to flow through the neuron.

the backward pass.

The approximation of choice is the fast sigmoid function, so named as it is faster to compute than the standard sigmoid function. The function and its derivative are plotted in Figure 5.5. We find that our model fails to converge without the use of a surrogate gradient.

5.2.3 Loss Calculation

The SLiNet computation graph can be unrolled exactly like a recurrent neural network (RNN), so backpropagation through time (BPTT) is used during training. Specifics of the unrolled computation graph can be found in Figure B.2. The SLiNet can also be encouraged to reach correct outputs at earlier timesteps by collecting the output from

each timestep, calculating the loss at each timestep, and summing all of these losses together. This unfortunately requires more memory than our GPU has capacity for, so we could not experiment with this method. Because our model required a relatively low number of timesteps to run, we utilized BPTT and looked back through all 20 timesteps.

5.3 On Converting an ANN

The ideas from this chapter are relevant only if one is training a SLiNet from scratch, which is the approach taken in this thesis. However, we did want to compare the performance of our trained SLiNet to that of one converted from an ANN. Inspired by Rathi et al. (2020), we tried an approach where we first scaled the weights and biases as an initialization step. We then trained this model on the data using the fast sigmoid surrogate gradient. Unfortunately, this model failed to converge. Hence, we cannot report on differences between training and converting a SLiNet on our object tracking task.

CHAPTER 6

Experiments and Results

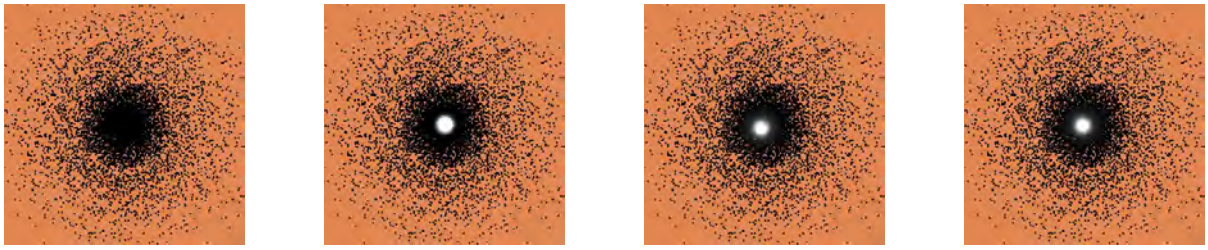
We test the operation of our foveation SLiNet in two ways. First, we move the target in three different trajectories and observe if the eye can track it. Second, we use data collected from human subjects to see if the eye movements resulting from the use of our SLiNet are realistic. Both types of tests are used in the work of Nakada et al. (2018).

We compare results using both the ONV and the D-ONV, with LiNet performance treated as a baseline. In each subsection, we plot the angular displacement of the ball with a black line and compare that to the movement generated by our 4-layer SLiNet (blue line) and a 5-layer LiNet (orange line).

6.1 Movements

6.1.1 Fixation

This test does not involve any movement of the target. We keep the target fixed in a location directly in front of the eye and observe how well it is kept fixated in the foveal region of the retina. An unrealistic model would fixate perfectly on the ball and not move, whereas a more realistic model allows the target to drift around the foveal region. These small movements are similar to micro-saccades in human eyes where a movement in one direction is balanced with a consecutive movement in the opposite direction. Figure 6.1 shows this type of movement generated from the use of our SLiNet using the D-ONV across four timesteps.



(a) (b) (c) (d)

Figure 6.1: In (a), the ball has just entered the eye's field of view. In (b) the eye has fixated on the ball. In (c)-(d) we see micro-saccades in opposite directions, with a slight movement in (c) and a correction to reach the position in (d).

6.1.2 Smooth Pursuit

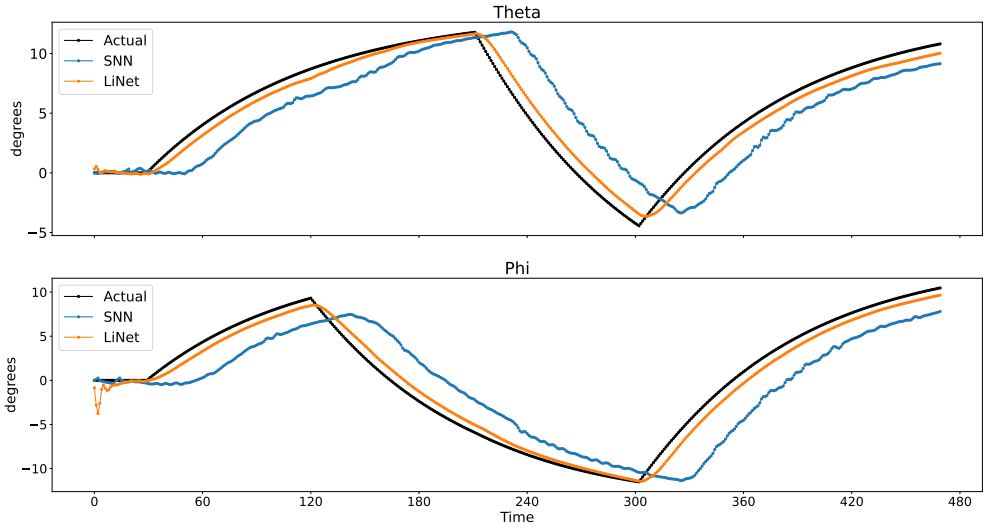
The smooth pursuit test moves the target slowly in both the horizontal θ and vertical ϕ directions. We observe if our SLiNet can successfully track this motion in both directions at the same time; the results are summarized in Figure 6.2.

In Figure 6.2a, we compare the performance of the models when using the ONV. The LiNet performs much better here, tracking the ball almost perfectly. The eye still successfully tracks the target with the SLiNet, but it fixates on a position a few degrees off from the center of the ball. In Figure 6.2b, we plot the eye gaze that results from using the D-ONV. Here, the LiNet struggles to keep the target in the center of its field of view while the SLiNet follows its motion almost perfectly. However, the motion that results from using the SLiNet is also more noisy. This is a result of micro-saccadic motion, where the eye moves slightly in one direction and then moves in the opposite direction in the next movement to compensate.

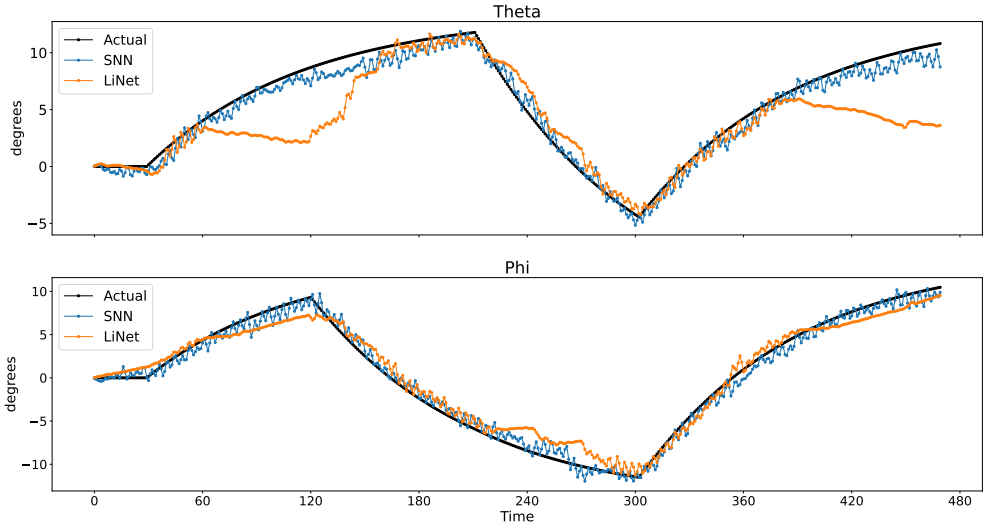
Our SLiNet, albeit noisy, tracks the target successfully using either the ONV or D-ONV.

6.1.3 Saccade

In a real saccade, the eye moves rapidly to fixate on a new location. To re-create this movement, we allow the eye to fixate on the ball and then rapidly move it to a new point within the eye's field of view. The results are summarized in Figure 6.3. Note that the



(a)



(b)

Figure 6.2: We compare the smooth movement of the target with the eye's gaze direction using different foveation networks. (a) The LiNet performs better using the ONV, but the SLiNet also keeps the target roughly in the center. (b) The SLiNet is far better at tracking the target when using a D-ONV.

seemingly instantaneous jumps in the black line represent saccadic movement where the ball rapidly moves to a new location in the eye's field of view.

In Figure 6.3a, we input an ONV to our two models. They exhibit similar performances, successfully tracking the target and keeping it focused on the center of the retina. When using the D-ONV, shown in Figure 6.3b, we see a difference between the two models. The SLiNet tracks the target better, exhibiting the same noisy motions as in the smooth motion test; however, the LiNet causes the eye to drift away from the target after about 4 consecutive saccadic motions.

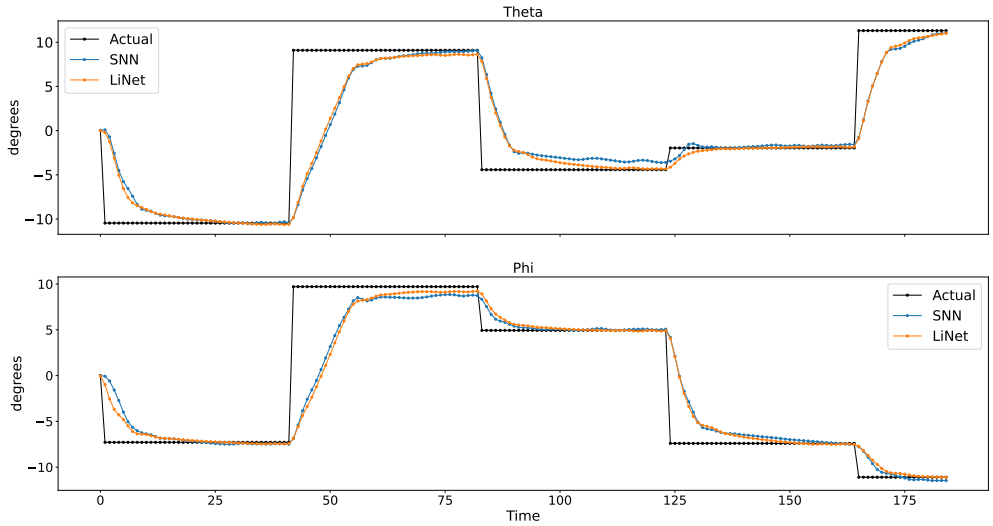
Our SLiNet, albeit noisy, tracks the target successfully. It also keeps the target in the center of the fovea when using the D-ONV, unlike the LiNet.

6.2 Comparison with Human Eye Movement

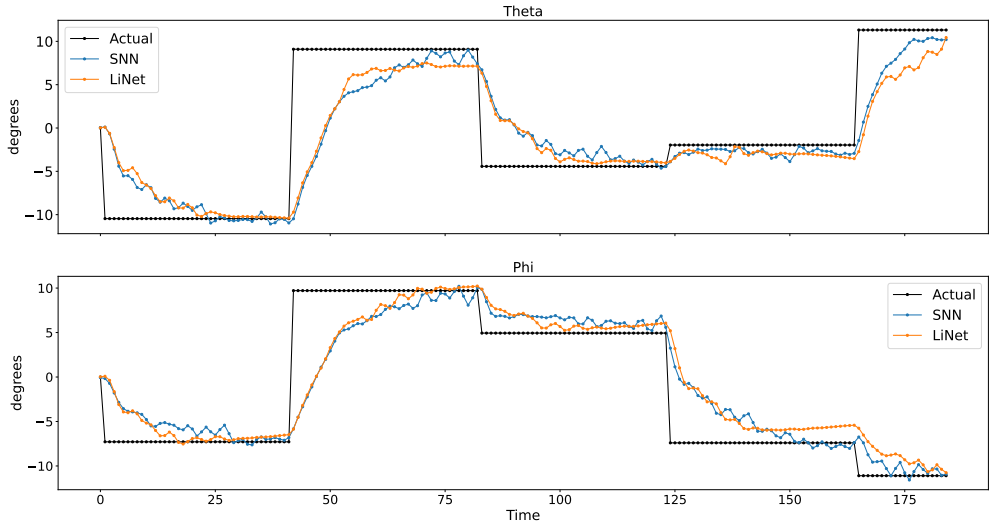
In this section we compare the angular displacement, velocity, and acceleration of our eye model to those of real human subjects. We use two movement patterns for the target to do so. The human recordings for smooth movement come from the work of Schraa-Tam et al. (2008), while the recordings for saccadic movement come from Thomas (1969).

6.2.1 Smooth Lateral Movement

We compare the angular displacements that result from moving our target back and forth in the horizontal direction. As seen in Figure 6.4, this results in a sinusoidal curve. Both models successfully track the ball when using the ONV, as seen in Figure 6.4b. The motions are relatively smooth, whereas the curve from the human subject in Figure 6.4a includes noisy movement. Our SLiNet produces these noisy movements when using the D-ONV, shown in Figure 6.4c. The LiNet's motion is not realistic and causes the eye to drift away.

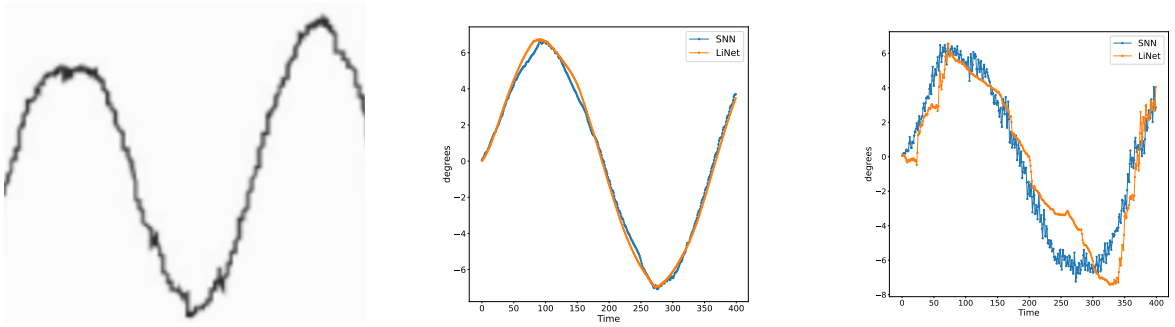


(a)



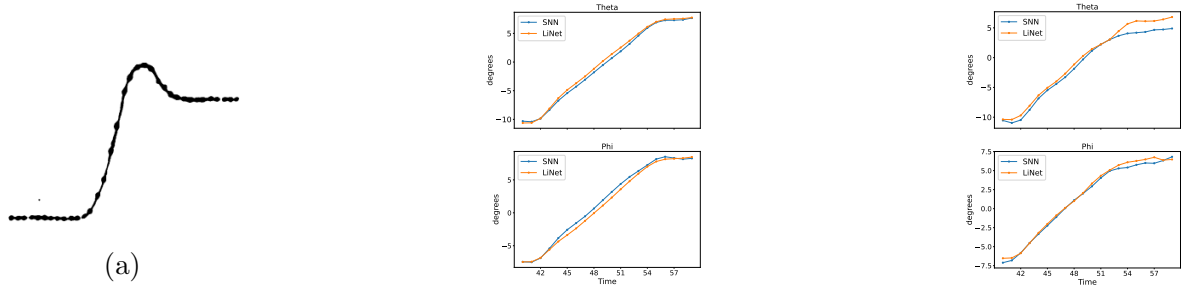
(b)

Figure 6.3: We compare the saccadic movement of the target with the center of the eye's gaze using the SLiNet and LiNet foveation networks. (a) Both models have similar performance, and are able to track the ball accurately. (b) The LiNet drifts away from the target after the fourth saccadic movement, whereas the SLiNet successfully tracks the target with a slightly noisy movement.



(a) (b) (c)

Figure 6.4: Comparison of angular displacements on horizontal motion. (a) Human. (b) ONV input. (c) D-ONV input.



(a) (b) (c)

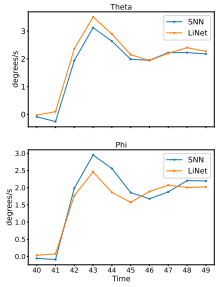
Figure 6.5: Comparison of angular displacements on saccadic motion. (a) Human. (b) ONV input. (c) D-ONV input.

6.2.2 Saccade

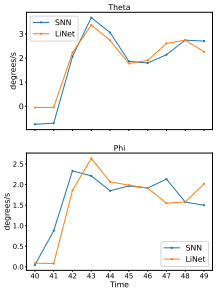
We again use saccadic motion, but this time note the angular velocity and acceleration of our eye model's movements in addition to the eye gaze direction. We compare these movements to those of a real human subject.

When comparing angular velocities in Figure 6.6 and angular accelerations in Figure 6.7, we see that our SLiNet produces very similar curves to those of the human subject. The D-ONV traces deviate from the human traces slightly, but they have similar characteristics. However, the angular displacements in Figure 6.5 are not as close to that of the human subject. There is still a slight "bump" in all of the curves, but not as pronounced as we see in the human's. It is important to note that the human curves do not have reported

(a)



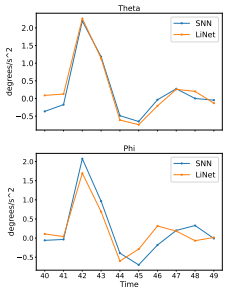
(b)



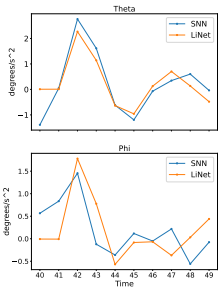
(c)

Figure 6.6: Comparison of angular velocities on saccadic motion. (a) Human. (b) ONV input. (c) D-ONV input.

(a)



(b)



(c)

Figure 6.7: Comparison of angular accelerations on saccadic motion. (a) Human. (b) ONV input. (c) D-ONV input.

amplitudes.

6.3 Spiking Analysis

Unfortunately, we did not have access to neuromorphic hardware to validate the low-power benefit claims of SLiNets. Instead, we analyze the sparsity of computation in a SLiNet versus the LiNet. We look at the number of neurons in each layer that have a non-zero activation in the LiNet and the nuber of neurons that emit at least one spike in the SLiNet.

Running our networks using the ONV examples from Figure 4.4a and Figure 4.4b, we see a similar amount of activated neurons in the LiNet and SLiNet. However, with

Layer	Number of Neurons	LiNet Activations (%)	SLiNet Activations (%)
1	8,640	32	2.4
2	1,728	21	7
3	345	33	17
4	69	39	45
5	13	31	-

Table 6.1: Comparing the percentage of neurons activated in each layer of the LiNet and the SLiNet when using the ONV from Figure 4.4c as input. An activated neuron in the LiNet outputs a non-zero value while an activated neuron in the SLiNet outputs at least one spike in the simulated time interval. There are significantly fewer activated neurons in the first few layers when using the SLiNet, thus demonstrating the sparse computation that results from using the D-ONV and spiking neurons.

the D-ONV from Figure 4.4c, the SLiNet uses much fewer neurons in the first few layers, which contain most of them. We summarize these results in Table 6.1 and present the membrane voltages over time for a subset of neurons from the first layer of our SLiNet in Figure 6.8.

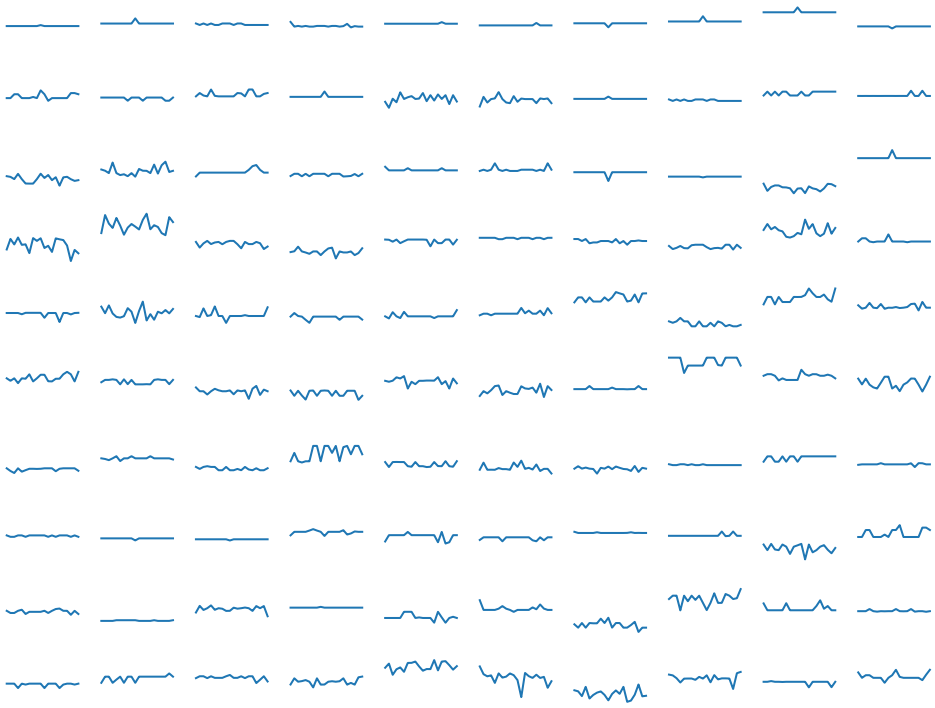


Figure 6.8: A plot of membrane voltages over time for a subset of 100 neurons from the first layer of our SLiNet, processing the ONV from Figure 4.4c. We see consistent spiking patterns from certain neurons, similar to real neuron spike trains. Many neurons are also not excited and have relatively flat voltage traces.

CHAPTER 7

Conclusion

This thesis has made two significant additions to the biomimetic eye model of Nakada et al. (2018); Lakshmipathi (2018); Nakada et al. (2021). First, we replaced the foveation LiNet with a more biologically plausible spiking neural network (SNN). Second, we trained our SLiNet on event-based data, allowing the eye to detect changes in the scene when performing visual tracking. In addition, our spike encoding method emulates the processing done by neurons at the retinal level and allows for sparse computation when compared to traditional ANNs. Finally, backpropagation training an SNN from scratch to solve a regression task was a novel achievement.

7.1 Discussion

Across all of our tests, the SLiNet was successfully able to track the target when using either the ONV or the D-ONV. The LiNet did not perform well with the D-ONV as input. We note, however, that the use of a D-ONV with our SLiNet resulted in noisier eye movements. This makes sense intuitively, as small movements are utilized to generate events that the eye can process to confirm where the ball is. Without these micro-saccadic movements, there would be fewer input events and the eye would likely struggle to track the ball.

Our SLiNet did not perfectly match the training performance of the LiNet. The LiNet converges very quickly and achieves lower training and validation loss values. However, our SLiNet still converges to a usable state. The LiNet's relatively large validation loss was an indication that it would struggle while using the D-ONV.

Additionally, our SLiNet was able to run using a relatively low number of timesteps. This is likely tied to the use of a gain factor in our rate encoding scheme, which encourages more neurons to spike more frequently.

Training our SLiNet from scratch allows for the use of more natural spike encoding methods when compared to converted SNNs. However, we were not able to comapare our model's performance with that of a converted SNN. Nevertheless, we believe that our work lays the groundwork for exploring different learning mechanisms, which is more general than tuning network parameters for a specific task.

7.2 Future Work

Many learning techniques are being studied outside of standard backpropagation. This includes spike time dependent plasticity (STDP), which is inspired by a theory of how neurons in the brain reinforce certain pathways (Diehl and Cook, 2015). More experimental network layers, such as neuron ensembles with inhibition (Bekolay et al., 2014), can also inspire more biologically plausible networks and help identify how certain parts of the visual cortex work.

We are also interested in exploring why latency encoding was unusable with our model. It potentially offers significant power savings over rate encoding and would be a valuable topic for future research.

Related to the biomechanical human musculoskeletal model of Nakada et al. (2018), we would like to explore the results of using two SLiNet-controlled eyes. The ONV as it stands is still a rather gross approximation in binocular vision, since the nervous system splits input from each eye into left and right regions at the optic chiasm. Exploring new architectures to process the visual input in this manner may offer interesting solutions.

Finally, all the SNNs in our work were emulated using GPU hardware. Given access to neuromorphic hardware, we would like to verify the power and latency improvements proposed by our hybrid SNNs. We would also like to verify if our output spike interpretation

method scales to other regression problems.

APPENDIX A

The Biomimetic Eye Model

In this appendix we provide more detail about the biomimetic eye model of Nakada et al. (2019).

A.1 Ocular Organs and Muscles

Light enters the eye through the pupil, and the iris is the muscle that controls the amount of light that makes its way to the retina. The iris is controlled in the simulation by two muscles: the pupillary sphincter, which constricts the pupil, and the pupillary dilator, which opens up the pupil. The pupil constricts when there is a large amount of light and it dilates when there is a low amount.

The cornea and lens serve to refract light to focus it onto the retina. Similar to the iris, the lens lengthens and shortens. The lens lengthens to focus on more distant objects and shortens to focus on closer objects.

Three pairs of extraocular (EO) muscles work together to move the eyeball with 3 degrees of freedom. One pair controls horizontal movement, one pair controls vertical movement, and the last pair creates a twisting motion.

All the muscles are simulated as Hill-type models.

A.2 Oculomotor Control System

A diagram of the oculomotor control system is shown in Figure A.1.

A single muscle activation signal dilates and constricts the pupil. This activation

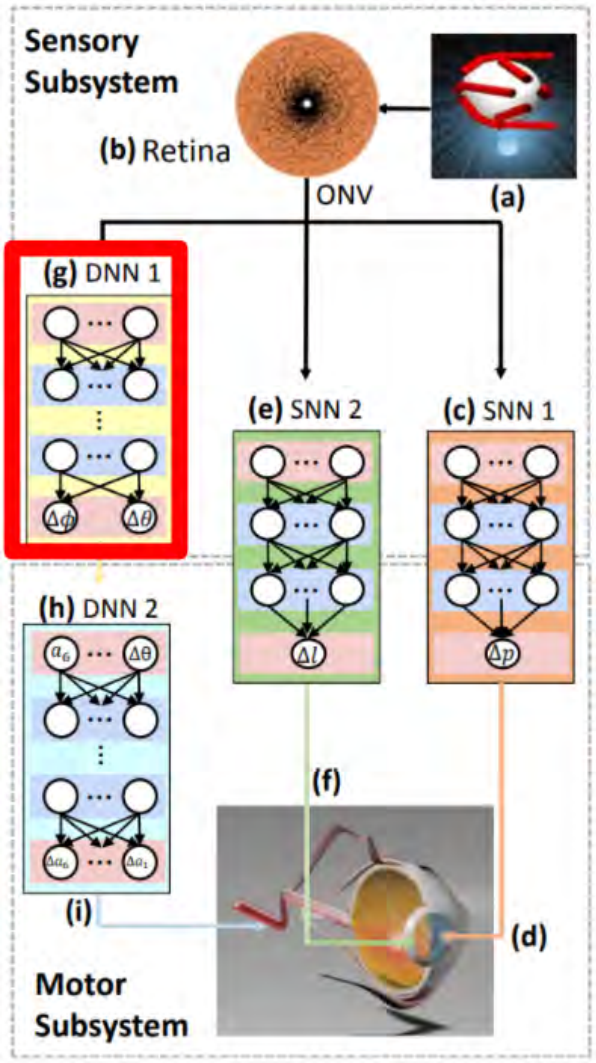


Figure A.1: Diagram of the eye's oculomotor control system. Note that SNN in this figure stands for "shallow neural network", not spiking neural network, and DNN stands for "deep neural network." The system is comprised of a sensory subsystem (top) and a motor subsystem (bottom). Using ray-tracing (a), rays are cast from the positions of photoreceptors on the retina (b) into the 3D scene, from which a 43,200 dimensional ONV of photoreceptor responses is computed. SNN 1 (c) and SNN 2 (e) input the ONV and output pupil (d) and lens (f) muscle activations responsible for luminance and focal accommodation, respectively. The ONV is also input to the foveation DNN 1 (highlighted in red), which is implemented as a LiNet, and which we replace with an SNN in this thesis. It outputs gaze angle changes, $\Delta\theta$ and $\Delta\phi$, required to track a moving visual target under observation within the field of view. These are input to the neuromuscular DNN 2 (h), which outputs the activation signals that drive the EO muscles (i) to produce the required eye movement. Diagram adapted from (Nakada et al., 2019).

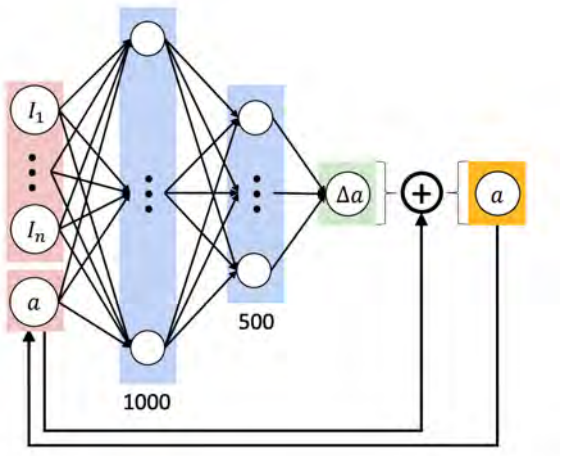


Figure A.2: The shallow, fully connected neural network architecture (labeled SNN 1 and SNN 2 in Figure A.1). Used to control the pupil and the lens. I_1 to I_n represent the ONV intensities that are input to the network. A change in muscle activation is output, which is added to the current activation value and passed back as input for the next timestep. Due to this connection the network is recurrent. Diagram from (Nakada et al., 2019).

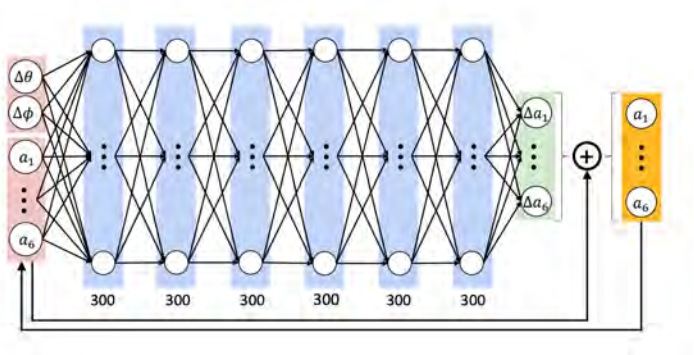


Figure A.3: The 6 layer, fully connected neural network used to control the EO muscles (labeled DNN 2 in Figure A.1). The current muscle activation values and angular displacement of the target are input to the network. Like the controller depicted in Figure A.2, this controller outputs changes in muscle activations and is recurrent. Diagram from (Nakada et al., 2019).

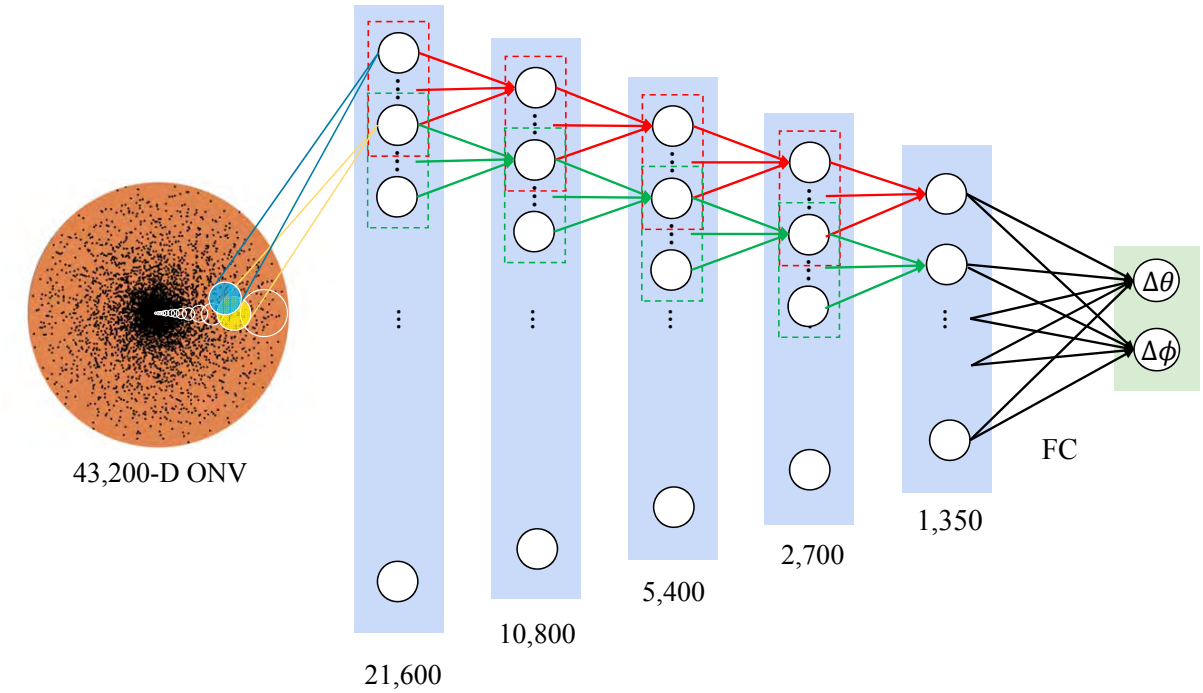


Figure A.4: Foveation DNN architecture. A LiNet backbone is followed by a fully-connected layer that outputs $\Delta\theta$ and $\Delta\phi$. Diagram from (Nakada et al., 2019).

is provided by a shallow fully connected neural network whose architecture is shown in Figure A.2. Unlike the iris, the lens is modeled with damped springs. It uses the same shallow neural network architecture as the iris and uses one activation value to control lens curvature. The neural network that controls the EO muscles is a deep, fully connected network, shown in Figure A.3. Like the other muscle controllers it outputs an activation value for each of the six muscles inducing them to contract. The neural networks are implemented as fully connected, recurrent models. Finally, the foveation DNN is illustrated in Figure A.4.

APPENDIX B

Mathematics of SNNs

To implement an SNN, we use snnTorch (Eshraghian et al., 2021). The main way that SNNs differ from conventional ANNs is that the activation function only outputs either a 1 (a "spike") or a 0 (no spike) and that inputs vary over time. Neurons maintain an internal voltage that increases when their inputs spike and decays in the absence of input spikes. These changes allow SNNs to replace floating point multiplications with simple additions because synapse weights are only multiplied by 1's or 0's.

B.1 Circuit Model of a Spiking Neuron

The fundamental unit of an SNN is the leaky integrate-and-fire (LIF) neuron. It can be represented as an RC circuit, as shown in Figure B.1. From the circuit, using Kirchoff's current law, we obtain

$$I_{\text{in}}(t) = I_R + I_C. \tag{B.1}$$

Next, we derive equations for I_R and I_C by defining V, the voltage across the resistor, and Q, the charge on the capacitor. Using Ohm's law, $I = V/R$, and the relationship $Q = CU_{\text{mem}}(t)$, we write equations for the resistor,

$$I_R(t) = \frac{U_{\text{mem}}(t)}{R}, \tag{B.2}$$

and the capacitor,

$$I_C(t) = \frac{dQ}{dt} = C\frac{dU_{\text{mem}}(t)}{dt}. \tag{B.3}$$

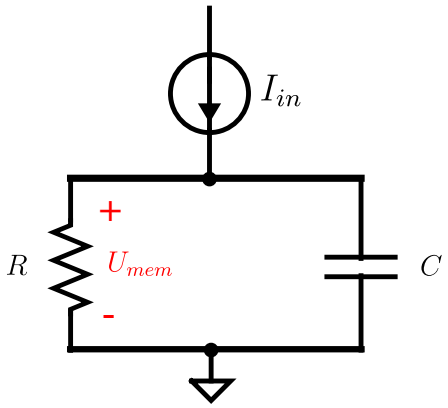


Figure B.1: The RC circuit representation of a leaky integrate and fire neuron.

Placing these definitions into our original equation, we obtain

$$I_{\text{in}}(t) = \frac{U_{\text{mem}}(t)}{R} + C\frac{dU_{\text{mem}}(t)}{dt} \tag{B.4}$$

and

$$RC\frac{dU_{\text{mem}}(t)}{dt} = -U_{\text{mem}}(t) + RI_{\text{in}}(t). \tag{B.5}$$

The units of the RHS are in voltage, while the term $\frac{dU_{\text{mem}}(t)}{dt}$ is in units of voltage/time. Therefore, the units of RC must be in time, and we refer to this as the time constant τ. This is a standard ordinary differential equation. Solving it analytically to determine that $U_{\text{mem}}(t) = U_0 e^{\frac{-t}{\tau}}$ would not be useful in a discrete-time neural network. Instead, starting from

$$\tau\frac{dU(t)}{dt} = -U(t) + RI_{\text{in}}(t), \tag{B.6}$$

we use the definition of the derivative to write

$$\tau\frac{U(t+\Delta t) - U(t)}{\Delta t} \approx -U(t) + RI_{\text{in}}(t) \tag{B.7}$$

and, ultimately,

$$U(t+\Delta t) \approx U(t) + \frac{\Delta t}{\tau}(-U(t) + RI_{\text{in}}(t)). \tag{B.8}$$

With the above, we achieve the desired neuron model with a membrane potential that increases with input current and decays in the absence of any input. The equation involves many hyperparameters, which would be difficult to tune. Therefore, the snnTorch package simplifies the equation as follows:

$$U(t + \Delta t) = (1 - \frac{\Delta t}{\tau})U(t) + \frac{\Delta t}{\tau} R I_{\text{in}}(t). \tag{B.9}$$

We remove a term by assuming that $I_{\text{in}}(t) = 0$, as this input current will be replaced by the presynaptic inputs to the neuron, thus obtaining

$$U(t + \Delta t) = (1 - \frac{\Delta t}{\tau})U(t). \tag{B.10}$$

Next, we denote the decay rate $(1 - \frac{\Delta t}{\tau}) = \beta$ (with $\Delta t << \tau$ for reasonable accuracy) to write

$$U(t + \Delta t) = \beta U(t). \tag{B.11}$$

Because we want to work with discrete timesteps, we can assume $\Delta t = 1$. We also assume that $R = 1$ in order to reduce the number of hyperparameters. Thus, we have the usable equation

$$U(t + 1) = \beta U(t) + WX(t + 1), \tag{B.12}$$

where W is a learnable parameter that weighs input spikes X. With $S(t)$ as our spiking function, we add a term to reset the membrane voltage when a neuron outputs a spike, and our final equation is

$$U[t + 1] = \underbrace{\beta U(t)}_{\text{decay}} + \underbrace{WX(t + 1)}_{\text{input}} - \underbrace{S(t)U_T}_{\text{reset}}, \tag{B.13}$$

with

$$S[t] = \begin{cases} 1 & \text{if } U(t) > U_T, \\ 0 & \text{otherwise.} \end{cases} \tag{B.14}$$

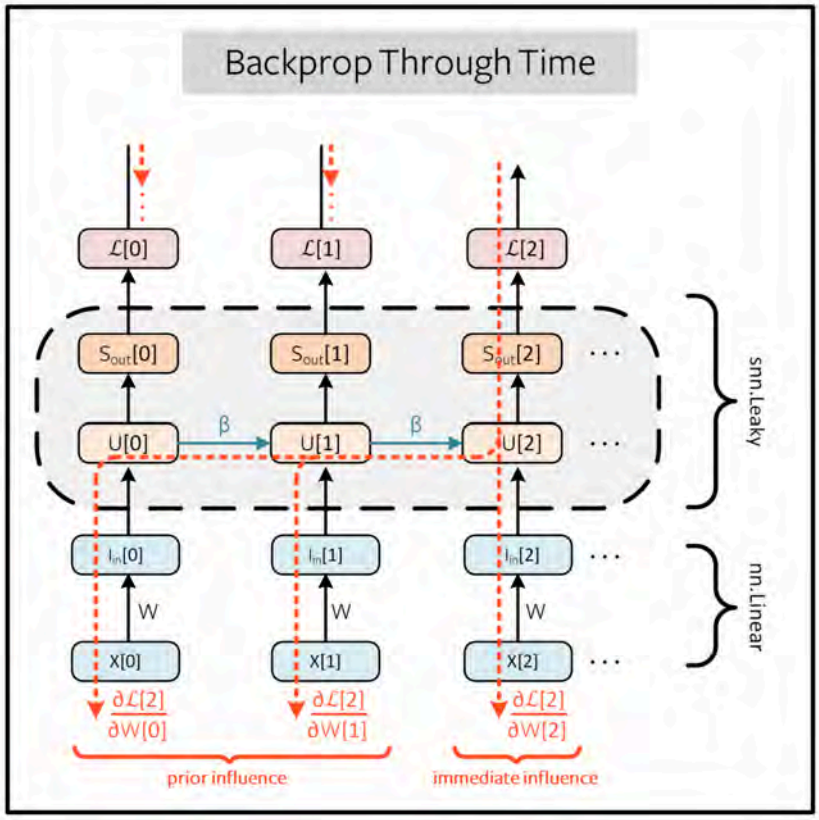


Figure B.2: Computation graph for an SNN, similar to an RNN. $X(t)$ is the time-varying, spiking input. After multiplication by the weights, the spikes become input current to the neuron. The membrane voltage $U(t)$ and spiking activation $S(t)$ functions are derived in Section B.1. Finally, the loss values $\mathcal{L}$ are computed at each timestep. Diagram from (Eshraghian et al., 2021).

B.2 Loop Unroll

From a computational graph perspective, SNNs are very similar to recurrent neural networks (RNNs), and we can use backpropagation through time (BPTT) to train our networks. The unrolled computation graph is shown in Figure B.2.

B.3 Neuron Parameters

We detail more neuron design choices regarding the type of spiking neuron that we use in our work:

Inhibition is an interesting option. In real neurons, the activation of one neuron can

inhibit other neurons from firing. Our more traditional architecture would have very sparse spiking with this type of learning enabled, so we do not use it. However, spiking RNNs may benefit from this feature.

Neurons can be distinguished by what is known as their order. A second-order neuron accounts for the fact that when a presynaptic neuron fires, it takes time for the signal to reach the current neuron. This is accounted for by adding a second hyperparameter α. These models are more complex to train and resulted in higher loss values, so we use first-order spiking neurons in our work.

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T., Rasmussen, D., Choo, X., Voelker, A., and Eliasmith, C. (2014). Nengo: A Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, 7(48):1–13. 42

Bouvier, M., Valentian, A., Mesquida, T., Rummens, F., Reyboz, M., Vianello, E., and Beigne, E. (2019). Spiking neural networks hardware implementations and challenges: A survey. *ACM Journal on Emerging Technologies in Computing Systems*, 15(2). 1

Brandli, C., Berner, R., Yang, M., Liu, S.-C., and Delbruck, T. (2014). A 240 × 180 130 db 3 µs latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, 49(10):2333–2341. 5

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C.-K., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., Weng, Y.-H., Wild, A., Yang, Y., and Wang, H. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99. 5

Diehl, P. and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, 9. 7, 42

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. (2015). Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. 6

Eshraghian, J. K., Ward, M., Neftci, E., Wang, X., Lenz, G., Dwivedi, G., Bennamoun, M., Jeong, D. S., and Lu, W. D. (2021). Training spiking neural networks using lessons from deep learning. *arXiv preprint arXiv:2109.12894*. 18, 48, 51

Gehrig, M., Shrestha, S. B., Mouritzen, D., and Scaramuzza, D. (2020). Event-based angular velocity regression with spiking networks. *CoRR*, abs/2003.02790. 7

Jose, J. T., Amudha, J., and Sanjay, G. (2015). A survey on spiking neural networks in image processing. In El-Alfy, E.-S. M., Thampi, S. M., Takagi, H., Piramuthu, S., and Hanne, T., editors, *Advances in Intelligent Informatics*, pages 107–115, Cham. Springer International Publishing. 1

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, 99:56–67. 7

Kim, S., Park, S., Na, B., and Yoon, S. (2020). Spiking-yolo: Spiking neural network for energy-efficient object detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07):11270–11277. 7

Lakshmipathi, A. S. (2018). Biomimetic modeling of the eye and deep neuromuscular oculomotor control. Master's thesis, University of California, Los Angeles. 1, 3, 41

Le, Q. V., Jaitly, N., and Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. 7

Maqueda, A. I., Loquercio, A., Gallego, G., Garcia, N., and Scaramuzza, D. (2018). Event-based vision meets deep learning on steering prediction for self-driving cars. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7

Mueggler, E., Rebecq, H., Gallego, G., Delbruck, T., and Scaramuzza, D. (2017). The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam. *The International Journal of Robotics Research*, 36(2):142–149. 6

Nakada, M., Chen, H., Lakshmipathy, A., and Terzopoulos, D. (2021). Locally-connected, irregular deep neural networks for biomimetic active vision in a simulated human. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4465–4472. 2, 12, 25, 41

Nakada, M., Lakshmipathy, A., Chen, H., Ling, N., Zhou, T., and Terzopoulos, D. (2019). Biomimetic eye modeling & deep neuromuscular oculomotor control. *ACM Trans. Graph.*, 38(6). viii, 1, 2, 3, 9, 11, 12, 13, 14, 44, 45, 46, 47

Nakada, M., Zhou, T., Chen, H., Weiss, T., and Terzopoulos, D. (2018). Deep learning of biomimetic sensorimotor control for biomechanical human animation. *ACM Trans. Graph.*, 37(4). 1, 11, 32, 41, 42

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63. 29

Purves, D., Augustine, G., Fitzpatrick, D., et al. (2001). *Anatomical Distribution of Rods and Cones*. Sinauer Associates. 10

Rathi, N., Srinivasan, G., Panda, P., and Roy, K. (2020). Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. 31

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 11. 6

Rullen, R. and Thorpe, S. (2001). Rate coding versus temporal order coding: What the retinal ganglion cells tell the visual cortex. *Neural computation*, 13:1255–83. 2

Schraa-Tam, C., Lugt, A., Frens, M., Smits, M., Broekhoven, P., and Geest, J. (2008). An fmri study on smooth pursuit and fixation suppression of the optokinetic reflex using similar visual stimulation. *Experimental brain research. Experimentelle Hirnforschung. Expérimentation cérébrale*, 185:535–44. 35

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in Neuroscience*, 13. 7

Shirley, P. and Morley, R. K. (2003). *Realistic Ray Tracing*. A. K. Peters, Ltd., Natick, MA, USA, 2 edition. 10

Tayarani-Najaran, M.-H. and Schmuker, M. (2021). Event-based sensing and signal processing in the visual, auditory, and olfactory domain: A review. *Frontiers in Neural Circuits*, 15. 20

Thomas, J. G. (1969). The dynamics of small saccadic eye movements. *The Journal of Physiology*, 200(1):109–127. 35

Wang, L., Qiu, Y.-H., and Zeng, Y. (2016). Coding properties of three intrinsically distinct retinal ganglion cells under periodic stimuli: A computational study. *Frontiers in Computational Neuroscience*, 10. 20