B.Comp. Dissertation

Spiking Neural Networks

By

Kuan Sheng Yuan, Jethro

Department of Computer Science

School of Computing

National University of Singapore

2019/20

B.Comp. Dissertation

Spiking Neural Networks

By

Kuan Sheng Yuan, Jethro

Department of Computer Science

School of Computing

National University of Singapore

2019/20

Project No: H226080
Advisor: Dr. Harold Soh
Deliverables:
Report: 1 Volume

Abstract

Spiking Neural Networks are third-generation neural networks that are well positioned to supersede the state-of-the-art deep learning methods in problems that are temporal in nature. Difficulties in training spiking neural networks mean they see little use outside of academic research. We evaluate a state-of-the-art gradient-based spiking neural network architectures, and propose its application in two settings: (1) as an agent in a reinforcement learning environment, and (2) as a controller in a robotics setting. We train spiking neural agents on the OpenAI gym environment, as well as a classifier on a novel multi-modal event-based dataset. The spiking neural network agents outperform artificial neural networks with fewer parameters, and a desirable property of being able to make early predictions.

Keywords:
Spiking Neural Networks, Reinforcement Learning, Multi-modal Machine Learning

Implementation Software and Hardware:
Python 3, PyTorch

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Deep neural networks (DNNs) have seen widespread industrial adoption, across tasks such as machine translation, image recognition, and in recommender systems. The remarkable success of deep learning can be attributed to advancements in gradient-based training methods and model architectures, allowing larger and deeper models to be trained and deployed.

Spiking Neural Networks (SNNs) is a nascent area of study. SNNs use neuronal units called spiking neurons which closely mimic the biological neuron, communicating via sparse, discrete spikes. This allows SNNs deployed on neuromorphic hardware to achieve favorable properties such as low power consumption, and fast inference. SNNs are rarely seen outside of academic research, because they remain difficult to train and have yet to outperform DNNs.

This thesis is a study of spiking neural networks and their applications. We are motivated by recent advancements in gradient-based training methods for spiking neural networks (Shrestha & Orchard, 2018; Huh & Sejnowski, 2018; Neftci, Mostafa, & Zenke, 2019), which has brought about a resurgence of interest in the field.

1.2 Research Outline

In chapter 2, we first provide a comprehensive study on SNNs.

The central property of SNNs that we exploit is the following:

> Unlike DNNs, SNNs are temporal in nature: spiking neurons store as state their membrane potential, which changes over time.

This property of SNNs positions them to supersede state-of-the-art DNNs in problems that are temporal in nature.

In chapter 3, we evaluate the feasibility of spiking neural networks as agents in a reinforcement learning setting. Reinforcement learning episodes often have a temporal element, due to the interdependence between states and actions in the environment. We hypothesised that spiking neural agents are well suited in many reinforcement learning settings.

In chapter 4, we contribute an event-driven visual-tactile spiking-neural network (VT-SNN), which enables fast perception on two event-based sensors: the NeuTouch (W. W. Lee et al., 2019), and the Prophesee event camera. This is a joint project between our group, the Collaborative Learning & Adaptive Robots (CLeAR) group, and the AiSKIN research group, which contributed the NeuTouch neuromorphic touch sensor. Spiking neural networks are able to naturally handle the sparse and discrete event-based data.

Chapter 2

Background

This chapter provides background knowledge about spiking neural networks. It reviews the differences between deep neural networks, and spiking neural networks (2.1). It introduces the basics of spiking neural networks (2.2), and its benefits (2.3). Finally, we discuss how to train spiking neural networks (2.4), and give some future directions.

2.1 The Generations of Neural Networks

Neural network models can be classified into three generations, according to their computational units: perceptrons, non-linear units, and spiking neurons (Maass, 1997).

Perceptrons can be composed to produce a variety of models, including Boltzmann machines and Hopfield networks. Non-linear units are currently the most widely used computational unit, responsible for the explosion of progress in machine learning research, in particular, the success of deep learning. These units traditionally apply differentiable, non-linear activation functions such across a weighted sum of input values.

There are two reasons second-generation computational units have seen so much success. First, the computational power of these units is greater than that of first-generation neural networks. Networks built with second-generation computational units with one hidden layer are universal approximators for any continuous function with a compact domain and range (Cybenko, 1989). Second, networks built with these units are trainable with well-researched

gradient-based methods, such as backpropagation.

The third generation of neural networks use computational units called spiking neurons. Much like our biological neurons, spiking neurons are connected to each other at synapses, receiving incoming signals at the dendrites and sending spikes to other neurons via the axon. Each computational unit stores some state: in particular, it stores its membrane potential at any point in time. Rather than fire at each propagation cycle, these computational units fire only when their individual membrane potentials crosses its firing threshold. A simple spiking neuron model is given in **??**.

From this section onwards, we shall term second-generation neural networks Artificial Neural Networks (ANNs), and third-generation neural networks Spiking Neural Networks (SNNs).

2.2 A Spiking Neuron Model

In spiking neural networks, neurons exchange information via spikes, and the information received depends on:

Firing frequencies The relative timing of pre and post-synaptic spikes, and neuronal firing patterns

Identity of synapses used Which neurons are connected, whether their synapses are inhibitory or excitatory, and synaptic strength

Each neuron has a corresponding model that encapsulates its state: the current membrane potential. As with the mammalian brain, incoming spikes increase the value of membrane potential. The membrane potential eventually decays to resting potential in the absence of spikes. These dynamics are often captured via first-order differential equations. Here we define the Spike Response Model (SRM), a simple but widely-used model describing the momentary value of a neuron i.

We define for presynaptic neuron j, $\epsilon_{ij}(t) = u_i(t) - u_{\text{rest}}$. For a few input spikes, the

membrane potential responds roughly linearly to the input spikes:

$$u_i t = \sum_j \sum_f \epsilon_{ij}(t - t_j^{(f)}) + u_{\text{rest}} \tag{2.1}$$

SRM describes the membrane potential of neuron i as:

$$u_i t = \eta(t - \hat{t}_i) + \sum_j \sum_f \epsilon_{ij}(t - t_j^{(f)}) + u_{\text{rest}} \tag{2.2}$$

where $\hat{t}_i$ is the last firing time of neuron i.

We refer to moment when a given neuron emits an action potential as the firing time of that neuron. We denote the firing times of neuron i by $t_i^{(f)}$ where $f = 1, 2, \ldots$ is the label of the spike. Then we formally denote the spike train of a neuron i as the sequence of firing times:

$$S_i(t) = \sum_f \delta\left(t - t_i^{(f)}\right) \tag{2.3}$$

where $\delta(x)$ is the Dirac-delta function with $\delta(x) = 0$ for $x \neq 0$ and $\int_{-\infty}^{\infty} \delta(x)dx = 1$. Spikes are thus reduced to points in time.

2.3 Motivating Spiking Neural Networks

Since second-generation neural networks have excellent performance, why bother with spiking neural networks? In this section, we motivate spiking neural networks from various perspectives.

2.3.1 Information Encoding

To directly compare ANNs and SNNs, one can consider the real-valued outputs of ANNs to be the firing rate of a spiking neuron in steady state. In fact, such rate coding has been used to explain computational processes in the brain (Pfeiffer & Pfeil, 2018). Spiking neuron models encode information beyond the average firing rate: these models also utilize the relative timing between spikes (Gütig, 2014), or spike phases (in-phase or out-of-phase). These time-dependent codes are termed temporal codes, and play an important role in biology. First, research has shown that different actions are taken based on single spikes (Stemmler, 1996). Second, relying

on the average firing rate would greatly increase the latency of the brain, and our brain often requires decision-making long before several spikes are accumulated. It has also been successfully demonstrated that temporal coding achieves competitive empirical performance on classification tasks for both generated datasets, as well as image datasets like MNIST and CIFAR (Comsa et al., 2019).

2.3.2 Biological Plausibility

A faction of the machine learning and neurobiology community strives for emulation of the biological brain. There are several incompatibilities between ANNs and the current state of neurobiology that are not easily reconciliated.

First, neurons in ANNs communicate via continuous-valued activations. This is contrary to neurobiological research, which shows that communication between biological neurons communicate by broadcasting spike trains: trains of action potentials to downstream neurons. The spikes are to a first-order approximation of uniform amplitude, unlike the continuous-valued activations of ANNs.

Second, backpropagation as a learning procedure also presents incompatibilities with the biological brain (Tavanaei, Ghodrati, Kheradpisheh, Masquelier, & Maida, 2019). Consider the chain rule in backpropagation:

$$\delta_j^\mu = g'\left(a_j^\mu\right) \sum_k w_{kj} \delta_k^\mu \tag{2.4}$$

δ_j^μ and δ_k^μ denote the partial derivatives of the cost function for input pattern μ with respect to the net input to some arbitrary unit j or k. Unit j projects feed-forward connections to the set of units indexed by k. $g(\cdot)$ is the activation function applied to the net input of unit j, denoted a_j^μ, w_{kj} are the feedforward weights projecting from unit j to the set of units indexed by k.

The chain rule formulation presents two problems. First, the gradients $g'(\cdot)$ requires derivatives, but $g(\cdot)$ in spiking neurons is represented by sum of Dirac delta functions, for which derivatives do not exist. Second, the expression $\sum_k w_{kj}\delta_k^\mu$ uses feedforward weights in a feed-

back fashion. This mean that backpropagation is only possible in the presence of symmetric feedback weights, but these do not exist in the brain. In addition, during backpropagation the error assignment for each neuron is computed using non-local information.

2.3.3 Neuromorphic Hardware

In a traditional Von Neumann architecture, the logic core operates on data fetched sequentially from memory. In contrast, in neuromorphic chips both computation and memory are distributed across computational units that are connected via synapses. The neuronal architecture and parameters hence play a key role in information representation and define the computations that are performed.

It has also been observed that spike-trains in the mammalian brain are often sparse in time, suggesting that timing and relative timings of spikes encode large amounts of information. Neuromorphic chips implement this same sparse, low-precision communication protocol between neurons on the chip, and by offering the same asynchronous, event-based parallelism paradigm that the brain uses, are able to perform certain workloads with much less power than Von Neumann chips.

These integrated circuits are typically programmed with spiking neural networks. Examples of such chips include IBM's TrueNorth (Merolla et al., 2014) and Intel's Loihi (Davies et al., 2018). Because spiking neural networks have not yet been successfully trained on many tasks, neuromorphic chips has seen little practical use. These chips have only recently been successfully used in robotic navigation (Tang, Shah, & Michmizos, 2019), and solving graph problems by manual construction of the network graph (Severa, Parekh, Carlson, James, & Aimone, 2016).

2.4 Training Spiking Neural Networks

As explained in subsection 2.3.3, it is desirable to train spiking neural networks to perform arbitrary tasks, utilizing power-efficient neuromorphic chips that break the Von Neumann bottleneck. We classify the training strategies by their usage of gradients, and discuss certain optimization techniques.

2.4.1 Non-gradient based methods

Spiking neurons communicate via spikes, hence, unlike ANNs, gradients are non-existent. In addition, backpropagation is not biologically plausible (see subsection 2.3.2). This motivates the use of plasticity-based methods and evolutionary strategies for training SNNs.

One category of learning rules used in SNNs are local learning rules. These rules include Hebbian learning (neurons that fire together wire together), and its extension: the spike-timing-dependent-plasticity rule (STDP). Inspired by experiments in neuroscience, central to these learning rules is the theme that neuron spike ordering and their relative timings encode information. STDP adjusts the strength of connections between neurons using the relative timing of a neuron's output and its input potentials (hence, spike-timing dependent).

In machine learning terminology, the weights of the synapses are adjusted according to fixed rules for each training example. Each synapse is given a weight $0 \leq w \leq w_{\max}$, characterizing its strength, and its change depends on the exact moments t_{pre} of pre-synaptic spikes and t_{post} of post-synaptic spikes (Sboev, Vlasov, Rybka, & Serenko, 2018):

$$\Delta w = \begin{cases} -\alpha\lambda \cdot \exp\left(-\frac{t_{\text{pre}}-t_{\text{post}}}{\tau_-}\right), \text{if } t_{\text{pre}} - t_{\text{post}} > 0 \\ \lambda \cdot \exp\left(-\frac{t_{\text{post}}-t_{\text{pre}}}{\tau_+}\right), \text{if } t_{\text{pre}} - t_{\text{post}} < 0 \end{cases} \tag{2.5}$$

where τ_+ and τ_- are time constants. $\tau_+ = 16.8ms$ and $\tau_- = 33.7ms$ are reasonable approximations obtained experimentally.

There are several libraries like BindsNET (Hazan et al., 2018) that simulate SNNs on Von Neumann computers implementing these rules. Recent attempts have been made to combine Reinforcement Learning and STDP: both in solving RL problems (Hazan et al., 2018), and using the reinforcement learning framework to train SNN (Bing et al., 2019; C. Lee, Panda, Srinivasan, & Roy, 2018). However, SNNs trained using the STDP learning rule have yet to achieve comparable performance compared to ANNs on relatively simple datasets like MNIST (Tavanaei et al., 2019).

2.4.2 Gradient-based methods

Performance is important for practical applications, and gradient-based training methods such as backpropagation has shown competitive performance. It is thus desirable to train spiking neural networks with these gradient-based methods.

There are several problems with spike-compatible gradient-based methods. First, most of these methods cannot train neurons in the hidden layers: they can only train neurons at the final layer, that receive the desired target output pattern (Urbanczik & Senn, 2009; J. Lee, Delbruck, & Pfeiffer, 2016). Second, the discontinuous, binary nature of spiking output needs to be addressed. For example, SpikeProp approximates the membrane threshold function at a local area with a linear function, introducing gradients and computing the exact formulae for error backpropagation for synaptic weights and spike times (Bohte, Kok, & Poutré, 2000). Others have modified the threshold function with a gate function (Huh & Sejnowski, 2018), used the alpha transfer function to derive gradient update rules (Comsa et al., 2019), and approximate the dirac-delta spikes with a probability density function (Shrestha & Orchard, 2018).

Another approach is converting trained ANN models into SNNs (Rueckauer, Lungu, Hu, & Pfeiffer, 2016). Common ANN layers such as softmax, batch normalization and max-pooling layers have their corresponding spiking counterparts.

Equilibrium Propagation was recently proposed to solve the neurobiological incompatibilities of backpropagation (Scellier & Bengio, 2017). Because the gradients are defined only in terms of local perturbations, the synaptic updates correspond to the standard form of STDP. The propagated signal encodes the gradients of a well-defined objective function on energy-based models, where the goal is to minimize the energy of the model. To resolve the issue of communication using binary-valued signals, step-size annealing was used to train spiking neural networks with Equilibrium Propagation (O'Connor, Gavves, & Welling, 2019).

2.4.3 Future Research Areas

A nascent area is local learning on neuromorphic chips. Thus far spiking neural networks are simulated and trained before deployment on a neuromorphic chip. In Intel's Loihi chip, each core contains a learning engine that can update synaptic weights using the 4-bit microcode-programmed learning rules that are associated with that synapse. This opens up areas for online learning.

Chapter 3

Reinforcement learning in Spiking Neural Networks

To achieve the first milestone (see **??**), I begin this research project by conducting a survey on SNN architectures that use gradients and have code readily available. We have found 2 repositories:

1. ihmehimmeli (Comsa et al., 2019)
2. SLAYER (Shrestha & Orchard, 2018)

We establish baselines, and evaluate these models on the STMNIST dataset.

It must be noted that the small number of samples in this dataset results in large error bars for each of the experiments conducted, and more samples must be collected before any conclusions are statistically significant. My experiments were run before the STMNIST dataset was updated. I have included my experimental results on this small dataset, as well as experimental results attained independently on the larger dataset for posterity.

3.1 Milestone 1: Supervised Learning on the STMNIST dataset

3.1.1 The STMNIST dataset

The STMNIST dataset is an unreleased dataset consisting of 200 samples, prepared by handwriting digits from 0 to 9, on a 10×10 tactile sensor grid [1]. The data is of dimension $(10, 10, t)$, where t is the number of time-steps for that particular sequence, t goes up to $20,000$. The values are either 0, -1 or 1, repesenting no spike, positive and negative spikes respectively. This dataset poses a few unique challenges:

1. Unlike existing spiking datasets like the NMNIST dataset (Serrano-Gotarredona & Linares-Barranco, 2015), the data is highly sparse: at each time-step, few neurons spike, which is consistent with spike trains that occur in the brain.

2. The dataset has an extremely large number of time-steps, which is a challenge for gradient-based methods such as backpropagation through time.

This dataset is designed for the supervised learning task of predicting the correct digit, given the spiking input.

3.1.2 Establishing Baselines

If we squash all time-steps together and binarize the values to 0s and 1s, we re-obtain the hand-drawn digit[2]. We can then train a MLP or CNN on the resultant image with regular supervised learning techniques. We use these trained models as our baselines. If it is indeed true that relative spike timings encode useful temporal information, then these baseline models discard all such temporal information, and should have weaker performance.

This method achieves a test accuracy of about 75%[3]. The AiSKIN team has independently trained a CNN on the updated, larger dataset, and obtained an accuracy of just above 80%.

[1] The STMNIST dataset preparation is a work-in-progress, and has grown to a sizable 7000 samples since I last worked with it.

[2] Colab notebook here.

[3] Colab notebook here.

3.1.3 Evaluating Ihmehimmeli

We encode the information temporally as described in the paper, by using the time to first spike for each cell in the 10 by 10 grid (Comsa et al., 2019)[4]. We evaluate their model with the same hyperparameters the authors used for the original MNIST problem, and achieved a similar accuracy of 75%, demonstrating that time-to-first-spike provides sufficient information for this classification task.

3.1.4 Evaluating SLAYER

Because we had decided that the STMNIST dataset was not able to provide conclusive results because of the small number of samples, we had given feedback to the AiSKIN team to collect more data, and opted not to run this experiment for SLAYER. I had shifted my research focus towards the reinforcement learning side of things after this decision.

SLAYER was eventually evaluated by other researchers in the lab group on the larger dataset, with reported test accuracies of 92.7%. This is significantly higher than the baseline CNN model trained by the AiSKIN team.

I summarise the evaluation results in Table 3.1.

Table 3.1: Test accuracies of models trained on the STMNIST dataset. Caveat: the results have no error-bars, communicated via word-of-mouth, and hyperparameters are not communicated. The objective of these runs are to get a rough idea of whether these models are performant: this is not the main evaluation task.

Dataset	Baseline CNN	ihmehimmeli	SLAYER
Initial	75%	75%	—
Updated	80%	—	92.7%

[4]Colab notebook here.

3.2 Milestone 2: Reinforcement Learning on the Cartpole Environment

3.2.1 The Cartpole Environment

We use the Cartpole-v0 environment, a popular environment provided by OpenAI. The official description is as follows (OpenAI, 2019):

> A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every time-step that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.
>
> CartPole-v0 defines "solving" as getting average reward of 195.0 over 100 consecutive trials.

3.2.2 Training a Simple ANN agent

First, we attempt to solve the problem with a simple MLP ANN agent that takes in the four observations, and outputs as logits the probability of each action. The action is then sampled from a 2-outcome categorical distribution with the logits as the outcome probabilities. We do this so we can compare the performance of the policies, learnt under the same conditions (learning rule, environment parameters etc.).

Since SLAYER uses PyTorch, we implement vanilla policy gradients in PyTorch to train the MLP agent. We use Sacred (Klaus Greff, Aaron Klein, Martin Chovanec, Frank Hutter, & Jürgen Schmidhuber, 2017) to log and ensure reproducibility of our experiments.[5]

Experiments show that the ANN model learns to solve the environment quickly when trained

[5]This repository is hosted on Github at `https://github.com/jethrokuan/snnrl/`. The repository is private, request access as required.

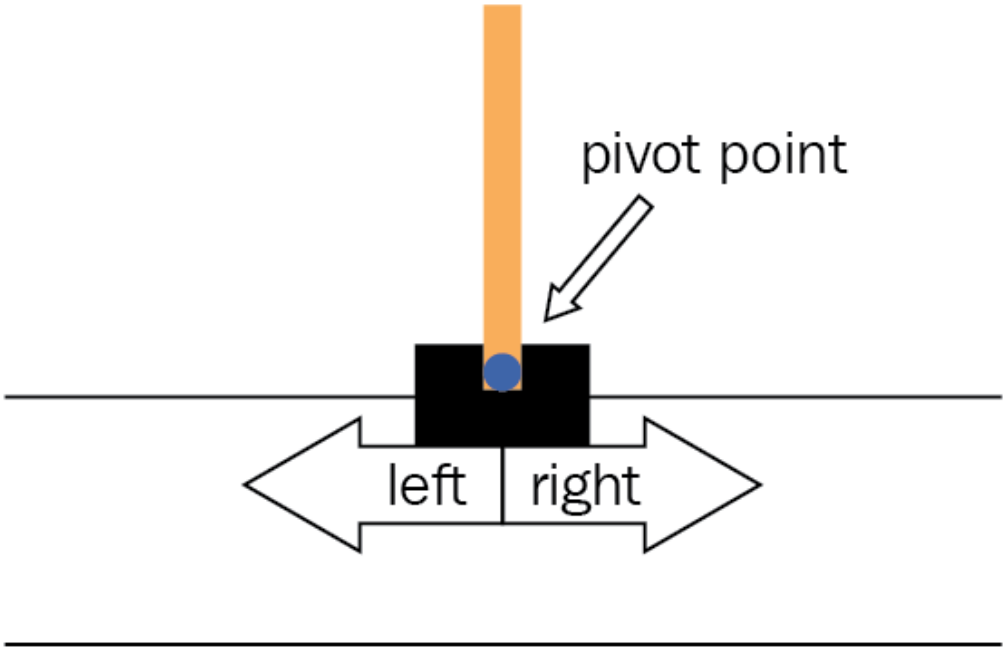

Figure 3.1: The OpenAI CartPole environment

with VPG, as shown in Figure 3.2. The agent obtains an average episodic return of 200, "solving" the environment at 2000 time-steps.

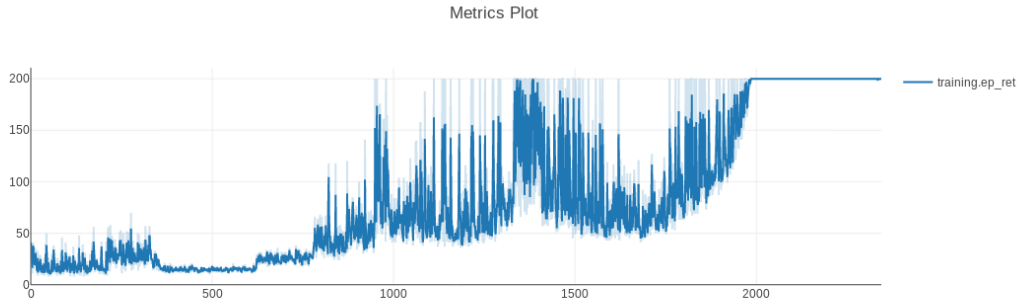

Figure 3.2: Plot of average episode returns over time-steps for a simple MLP agent.

3.2.3 Training a SNN agent

Similarly, we attempt to train the SLAYER model to solve the CartPole-v0. Each SLAYER layer takes in a spike train as input and produces a spike train as output. The objective function

for supervised learning tasks such as classification is defined in the paper, and is similar to cross-entropy (Shrestha & Orchard, 2018).

Since the input to the Slayer model are spike trains, we require an encoder mapping the environment observations into the fixed-length spike trains that SLAYER accepts,as shown in Figure 3.3.

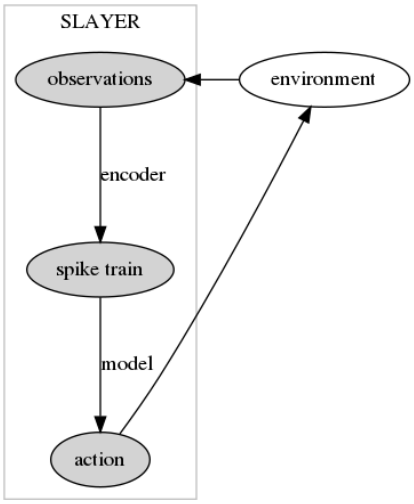

Figure 3.3: The SLAYER model requires an additional encoder to transform the observations into spike trains as input.

We encode the observations by modelling the observations as Poisson processes (Heeger, 2000). First, we convert each of the four observations into 2 non-negative observations: their absolute value, and their sign. A negative value has an observation value of 0, and a positive value has observation value of 1. We divide time into short, discrete intervals δt, and generate a sequence of random numbers $x[i]$ uniformly between 0 and 1. For each interval, if $x[i] \leq r\delta t$, generate a spike. We produce spike trains of 300 time-steps as input to the SLAYER model from the 8 observation values (scaled appropriately).

As shown in Figure 3.4, the SLAYER model fails to learn to solve the problem.

The problem likely resides in way encoding is done. The domain of the value of some observations in the Cartpole environment is the range over all real numbers, and encoding values over the entire range of real numbers is tricky. There is no linear mapping, and changes

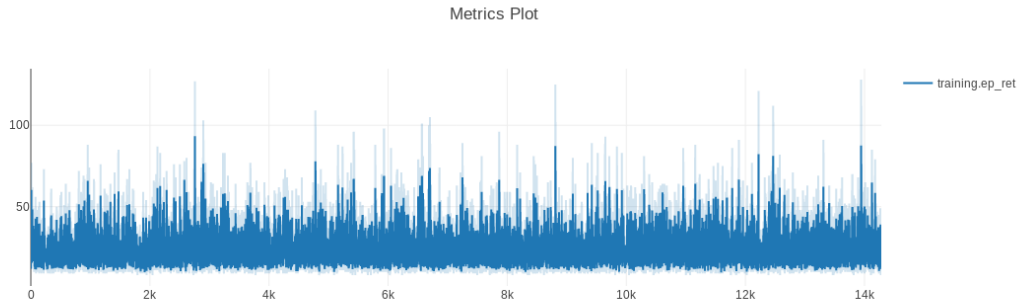

Figure 3.4: The SLAYER model with a Poisson encoder for the observations fails to learn to solve the problem.

in observation values may not result in changes of noticeable magnitude in the mapped value.

There has, however, been works on encoding images into spike trains. This is particularly common as many spiking neural architectures are evaluated on a converted MNIST image dataset. Hence, we move towards training spiking neural networks on image observations.

3.2.4 The ImageCartpole Environment

To bypass the encoding issue described earlier, we turn towards training agents on the pixels of the environment. We create a Cartpole environment such that the observation is the difference in pixels of the current frame and the previous frame. We term this modified environment ImageCartpole.

We train a simple CNN on this environment, using VPG. Training this agent takes significantly more time, as seen in Figure 3.5. However, the agent is still able to learn from the observations, suggesting that this environment is solvable by a SNN agent.

At the time of submission, the SNN agent is unable to learn a good policy, but more engineering work is required before any conclusions can be made.

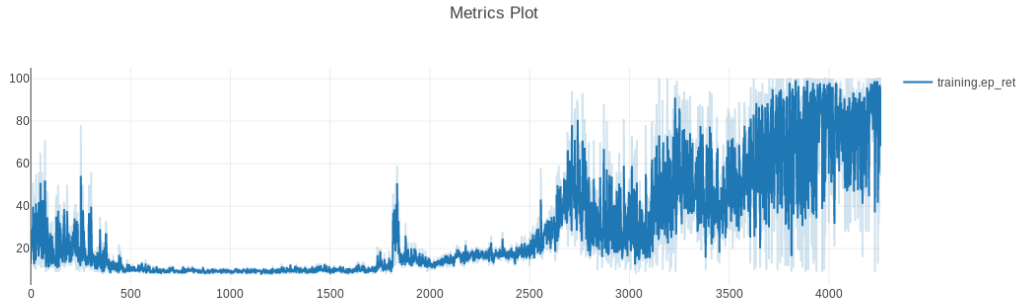

Figure 3.5: Plot of the average episode return of a CNN agent on the ImageCartpole environment over time.

Chapter 4

Event-driven Visual-Tactile Sensing with Spiking Neural Networks

Chapter 5

Future Work

It is hoped that the SLAYER model is able to learn to solve simple environments such as the Cartpole environment. Once it is established that gradient-based RL methods also work for training SLAYER, we would move on towards milestones 3 and 4. If we are unable to learn a good policy via existing reinforcement learning methods, some exploration into tweaking the learning rules or spiking neural architecture is warranted.

As a closing thought, the way the cartpole environment is currently being used also does not fully utilize the power of spiking neural networks. At each time-step, observations are converted to full spike trains. SNNs are often able to make predictions based on early spikes, albeit less reliable. This is not exploited in the current environment formulation, where a full spike train of fixed length is received before an action is taken.

In my idealized formulation of RL environment for SNNs, the following should be satisfied:

1. The action space should contain a no-op: in the case of the cartpole environment, this is as simple as adding the action of not applying force to the cart, instead of choosing between left or right

2. At each observation, the observation should be spikes, rather than spike trains

The SNN agent receives spikes at each time-step, and after gaining enough confidence to take an action, it takes the action at that time-step. I hypothesize that SNN agents should be able to solve such environments, by exploiting the temporal information encoded in the relative

timing between spikes (in this case, number of timesteps) and that each spiking neuron stores such temporal state, without the use of recurrent networks or replay buffers common in current ANN setups.

References

Bing, Z., Baumann, I., Jiang, Z., Huang, K., Cai, C., & Knoll, A. (2019). Supervised learning in snn via reward-modulated spike-timing-dependent plasticity for a target reaching vehicle. *Frontiers in Neurorobotics*, *13*, 18. Retrieved from `https://www.frontiersin.org/article/10.3389/fnbot.2019.00018` doi: 10.3389/fnbot.2019.00018

Bohte, S., Kok, J., & Poutré, J. (2000, 01). Spikeprop: backpropagation for networks of spiking neurons. In (Vol. 48, p. 419-424).

Comsa, I. M., Potempa, K., Versari, L., Fischbacher, T., Gesmundo, A., & Alakuijala, J. (2019). Temporal coding in spiking neural networks with alpha synaptic function. *CoRR*. Retrieved from `http://arxiv.org/abs/1907.13223v2`

Cybenko, G. (1989, Dec 01). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, *2*(4), 303–314. Retrieved from `https://doi.org/10.1007/BF02551274` doi: 10.1007/BF02551274

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., … others (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, *38*(1), 82–99.

Gütig, R. (2014). To spike, or when to spike? *Current Opinion in Neurobiology*, *25*(nil), 134-139. Retrieved from `https://doi.org/10.1016/j.conb.2014.01.004` doi: 10.1016/j.conb.2014.01.004

Hazan, H., Saunders, D. J., Khan, H., Patel, D., Sanghavi, D. T., Siegelmann, H. T., & Kozma, R. (2018). Bindsnet: A machine learning-oriented spiking neural networks library in python. *Frontiers in Neuroinformatics*, *12*, 89. Retrieved from

https://www.frontiersin.org/article/10.3389/fninf.2018.00089 doi: 10.3389/fninf.2018.00089

Heeger, D. (2000). Poisson model of spike generation. *Handout, University of Standford*, *5*, 1–13.

Huh, D., & Sejnowski, T. J. (2018). Gradient descent for spiking neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems 31* (pp. 1433–1443). Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/7417-gradient-descent-for-spiking-neural-networks.pdf

Klaus Greff, Aaron Klein, Martin Chovanec, Frank Hutter, & Jürgen Schmidhuber. (2017). The Sacred Infrastructure for Computational Research. In Katy Huff, David Lippa, Dillon Niederhut, & M. Pacer (Eds.), *Proceedings of the 16th Python in Science Conference* (p. 49 - 56). doi: 10.25080/shinma-7f4c6e7-008

Lee, C., Panda, P., Srinivasan, G., & Roy, K. (2018). Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Frontiers in Neuroscience*, *12*, 435. Retrieved from https://www.frontiersin.org/article/10.3389/fnins.2018.00435 doi: 10.3389/fnins.2018.00435

Lee, J., Delbruck, T., & Pfeiffer, M. (2016, 08). Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, *10*. doi: 10.3389/fnins.2016.00508

Lee, W. W., Tan, Y. J., Yao, H., Li, S., See, H. H., Hon, M., … Tee, B. C. K. (2019). A neuro-inspired artificial peripheral nervous system for scalable electronic skins. *Science Robotics*, *4*(32). Retrieved from https://robotics.sciencemag.org/content/4/32/eaax2198 doi: 10.1126/scirobotics.aax2198

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, *10*(9), 1659 - 1671. Retrieved from http://www.sciencedirect.com/science/article/pii/S0893608097000117 doi: https://doi.org/10.1016/S0893-6080(97)00011-7

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., …

Modha, D. S. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, *345*(6197), 668–673. Retrieved from https://science.sciencemag.org/content/345/6197/668 doi: 10.1126/science.1254642

Neftci, E. O., Mostafa, H., & Zenke, F. (2019). Surrogate gradient learning in spiking neural networks. *CoRR*. Retrieved from http://arxiv.org/abs/1901.09948v2

O'Connor, P., Gavves, E., & Welling, M. (2019, 16–18 Apr). Training a spiking neural network with equilibrium propagation. In K. Chaudhuri & M. Sugiyama (Eds.), *Proceedings of machine learning research* (Vol. 89, pp. 1516–1523). PMLR. Retrieved from http://proceedings.mlr.press/v89/o-connor19a.html

OpenAI. (2019). *Openai gym.* https://gym.openai.com/envs/CartPole-v0/. (Online; accessed 02 November 2019)

Pfeiffer, M., & Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Frontiers in neuroscience*, *12*.

Rueckauer, B., Lungu, I.-A., Hu, Y., & Pfeiffer, M. (2016). Theory and tools for the conversion of analog to spiking convolutional neural networks. *CoRR*. Retrieved from http://arxiv.org/abs/1612.04052v1

Sboev, A., Vlasov, D., Rybka, R., & Serenko, A. (2018). Spiking neural network reinforcement learning method based on temporal coding and stdp. *Procedia Computer Science*, *145*(nil), 458-463. Retrieved from https://doi.org/10.1016/j.procs.2018.11.107 doi: 10.1016/j.procs.2018.11.107

Scellier, B., & Bengio, Y. (2017). Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in Computational Neuroscience*, *11*, 24. Retrieved from https://www.frontiersin.org/article/10.3389/fncom.2017.00024 doi: 10.3389/fncom.2017.00024

Serrano-Gotarredona, T., & Linares-Barranco, B. (2015). Poker-dvs and mnist-dvs. their history, how they were made, and other details. *Frontiers in Neuroscience*, *9*, 481. Retrieved from https://www.frontiersin.org/article/10.3389/fnins.2015.00481 doi: 10.3389/fnins.2015.00481

Severa, W., Parekh, O., Carlson, K. D., James, C. D., & Aimone, J. B. (2016). Spiking network algorithms for scientific computing. *2016 IEEE International Conference on Rebooting Computing (ICRC)*, 1-8.

Shrestha, S. B., & Orchard, G. (2018). Slayer: Spike layer error reassignment in time. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems 31* (pp. 1412–1421). Curran Associates, Inc. Retrieved from `http://papers.nips.cc/paper/7415-slayer-spike-layer-error-reassignment-in-time.pdf`

Stemmler, M. (1996). A single spike suffices: the simplest form of stochastic resonance in model neurons. *Network: Computation in Neural Systems*, *7*(4), 687-716. Retrieved from `https://doi.org/10.1088/0954-898x_7_4_005` doi: 10.1088/0954-898x_7_4_005

Tang, G., Shah, A., & Michmizos, K. P. (2019). Spiking neural network on neuromorphic hardware for energy-efficient unidimensional slam. *CoRR*. Retrieved from `http://arxiv.org/abs/1903.02504v2`

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., & Maida, A. (2019). Deep learning in spiking neural networks. *Neural Networks*, *111*, 47 - 63. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0893608018303332` doi: https://doi.org/10.1016/j.neunet.2018.12.002

Urbanczik, R., & Senn, W. (2009). A gradient learning rule for the tempotron. *Neural Computation*, *21*(2), 340-352. Retrieved from `https://doi.org/10.1162/neco.2008.09-07-605` doi: 10.1162/neco.2008.09-07-605