B.Comp. Dissertation

Event-Driven Visual-Tactile Learning for Robots

By

Kuan Sheng Yuan, Jethro

Department of Computer Science

School of Computing

National University of Singapore

2019/20

B.Comp. Dissertation

Event-Driven Visual-Tactile Learning for Robots

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

In this work, we contribute an event-driven visual-tactile perception system, built on spiking neural networks. Our perception system is trained, and tested end-to-end on novel multi-modal event-based data for two sensor modalities: (1) Tactile data, from our biologically-inspired fingertip tactile sensor, NeuTouch and (2) visual data, from the Prophesee camera. We evaluate our system on two robotic tasks: container classification, and slip detection. On both tasks, we observe good classification accuracies relative to standard deep learning methods. Our system can be run on neuromorphic hardware, making this a crucial first step towards enabling power-efficient, intelligent robots.

Keywords:
Spiking Neural Networks, Multi-modal Machine Learning

Implementation Software and Hardware:
Python 3, PyTorch

Contents

Chapter 1

Introduction

Human beings are blessed with innate abilities to integrate sensory information from different stimuli. Our sense of smell, hearing, touch, vision, and taste each contribute to how we perceive and act in the world. Consider the scenario of fetching a carton of soy-milk from the fridge; humans use vision to locate the carton, and are also able to infer from grasping the object, the amount of soy-milk the carton contains. These actions (and inferences) are performed robustly using a power-efficient neural substrate — compared to popular deep learning approaches for using multiple sensor modalities in artificial systems, human brains require far less energy [1].

In this work, we take crucial steps towards efficient visual-tactile perception for robotic systems. We gain inspiration from biological systems, which are *asynchronous* and *event-driven*. In contrast to resource-hungry deep learning methods, event-driven perception offers an alternative approach that promises power-efficiency and low-latency — features that are ideal for real-time mobile robots. However, event-driven systems remain under-developed relative to standard synchronous perception methods [2].

We make multiple contributions that advance event-driven visual-tactile perception. First, to enable richer tactile sensing, we use the NeuTouch fingertip sensor. Compared to existing commercially-available tactile sensors, NeuTouch's neuromorphic design enables scaling to larger number of taxels while retaining low latencies.

Next, we investigate multi-modal (visual-tactile) learning using the NeuTouch and the Prophesee event-based camera. Specifically, we develop a *visual-tactile spiking neural network* (VT-

SNN) that incorporates both sensory modalities for supervised-learning tasks. Different from conventional deep artificial neural network (ANN) models, SNNs process discrete spikes asynchronously, and thus, are arguably better suited to the event data generated by our neuromorphic sensors. In addition, SNNs can be used on efficient low-power neuromorphic chips such as the Intel Loihi [3].

Our experiments center on two robot tasks: object classification and (rotational) slip detection. In the former, we tasked the robot to determine the type of container being handled and amount of liquid held within. The containers were opaque with differing stiffness, and hence, both visual and tactile sensing are relevant for accurate classification. We show that relatively small differences in weight (≈ 30g across 20 object-weight classes) can be distinguished by our prototype sensors and spiking models. Likewise, the slip detection experiment indicates rotational slip can be accurately detected within 0.08s (visual-tactile spikes processed every ≈ 1ms). In both experiments, SNNs achieved competitive (and sometimes superior) performance relative to ANNs with similar architecture.

The work presents an exciting opportunity to enable power-efficient intelligent robots. Presented with labeled data, an event-driven perception network can be trained end-to-end, and used on neuromorphic chips as robotic controllers.

1.1 Research Outline

In chapter 2, we first provide a comprehensive study on SNNs. We discuss why and when we should use spiking neural networks, and how to train them. Crucially, we draw similarities between spiking neural networks and recurrent neural networks, a recently discovered property that enables the end-to-end training of these networks using gradient-based methods.

In chapter 3, we contribute an event-driven visual-tactile spiking-neural network (VT-SNN), which enables fast perception on two event-based sensors: the NeuTouch [4], and the Prophesee event camera. Here, we detail the experimental setups for the robotic tasks: object classification and slippage detection. We discuss our experimental results, and demonstrate that spiking neural networks achieve competitive (and sometimes superior) performance over deep learning

methods, and in addition having the unique property of early classification.

This thesis was originally an exploratory project on spiking neural networks. In the duration of this research, several different research directions have been explored. In the chapters that follow, we discuss research directions we have taken that were later abandoned.

In chapter 4, we evaluate the feasibility of spiking neural networks as agents in a reinforcement learning setting. Reinforcement learning environments are often temporal in nature, due to the interdependence between states and actions in the environment. We hypothesised that spiking neural agents were well suited in many reinforcement learning settings.

1.2 Individual Contributions

The event-driven visual-tactile perception system is a joint work between our group, the Collaborative Learning & Adaptive Robots (CLeAR) group, and the TEE research group.

The TEE research group contributed the novel NeuTouch tactile sensor, which was used to collect data for the tactile modality. The CLeAR group contributed: (1) a visual-tactile spiking neural network (VT-SNN) that leverages multiple event sensor modalities, (2) systematic experiments demonstrating the effectiveness of our event-driven perception system on object classification and slip detection, with comparisons to conventional ANN methods, and (3) visual-tactile event sensor datasets comprising more than 50 different object classes across the experiments, including RGB images and proprioceptive data from the robot.

Among the contributions of the CLeAR group, my individual contributions include the experimentation (training and evaluation) of the VT-SNN models, as well as some guidance on the ANN models. I am also looking into running these models on neuromorphic hardware, to quantify the performance and power gains of using the spiking models. During the research process, I also maintained the code repository at high-quality. Model training was designed to be reproducible, and parameter searches could be done efficiently across multiple GPUs. Different experimental runs could also be plotted, and their results aggregated for later analyses.

All work in subsequent chapters () are my own.

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 (2.3), and its benefits (2.2). 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 [5]. Perceptrons can be composed to produce a variety of models, including Boltzmann machines and Hopfield networks. One characteristic feature of perceptrons is that only give digital output.

Non-linear units apply an activation function with a continuous set of possible output values to a weighted sum of the inputs. Classic examples of networks of this second generation include feed-forward and recurrent sigmoidal neural networks. These networks are able to compute functions with analog input and output. Non-linear units are currently the most widely used computational unit, and is responsible for the explosion of progress in machine learning research, in particular, the success of deep learning. This is largely because networks built with these units are trainable with well-researched gradient-based methods, such as backpropagation.

Name	Input	Output	Examples
(1) Perceptrons	Digital	Digital	Hopfield Networks, Boltzmann Machines
(2) Non-linear Units	Digital/Analog	Digital/Analog	Feed-forward & Recurrent ANNs
(3) Spiking Neurons	Analog	Analog	Feed-forward SNNs

Table 2.1: The three generations of neural networks.

Despite being biologically inspired, second-generation neural networks (ANNs) bear little resemblance to the human cortex. In contrast to the digital computation in ANNs, networks of neurons perform fast analog computations. This inspired a the third generation of neural networks use computational units called *spiking neurons*, which communicate via discrete spikes. 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 downstream neurons via the axon. Each computational unit stores its membrane potential, which fluctuates over time based on well-defined neuronal dynamics. Rather than firing at each propagation cycle, these computational units fire only when their individual membrane potentials crosses its firing threshold.

Henceforth, we shall term second-generation neural networks artificial neural networks (ANNs), and third-generation neural networks spiking neural networks (SNNs).

2.2 Motivating Spiking Neural Networks

Since ANNs have excellent performance, and can handle both digital and analog input and output, why should we bother with spiking neural networks? In this section, we motivate spiking neural networks from various perspectives.

2.2.1 Information Encoding via Spikes

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, this scheme, termed *rate coding*, has been used to explain computational processes in the brain [2].

However, experiments have shown that different actions are taken based on single spikes [6]. In addition, the humans are capable of performing tasks such as visual pattern analysis and pattern classification in 100ms [7]. Rate coding strategies are far too inefficient for the rapid information transmission required for sensory processing. In addition, the firing distribution of biological neurons is heavily skewed towards lower firing rates. To obtain a good estimate of the firing rate, many spikes would be required.

Different from ANNs, spiking neuron models are able to encode information beyond the average firing rate: these models also utilize the relative timing between spikes [8], or spike phases (in-phase or out-of-phase). These time-dependent codes are termed *temporal codes*, and play an important role in biology. 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 [9].

There is additional benefit to encoding information directly in the temporal domain. Architectures for atemporal artificial neural networks using a memory mechanism, such as the LSTM, have been proposed. However, in these networks, every neuron wait on the activation of all neurons in previous layers before producing an answer. In spiking neural networks, the output layers spike over time, and predictions can be made at any time-step. Hence, SNNs are able to operate in two regimes. The first regime is a highly accurate but slow regime, where predictions are made at later time-steps. This is the regime that atemporal ANNs support, where predictions are typically made only after a fixed time-step T. The second regime is a low accuracy but fast regime, where the network fast predictions, at a much lower accuracy. This behaviour mirrors the speed-accuracy tradeoff observed in human decision-making [9], and we also observe and exploit this property in our work described in chapter 3.

By observing that biological neurons use spikes to communicate information, a wide range of coding schemes have been developed. Each of these codes have different information capacities, and their pros and cons. We defer this discussion to [7], but summarize the coding schemes in Table 2.2. Spiking neural networks are able to take advantage of these various coding schemes.

Coding Scheme	Encoding of Information
Rate Coding	Firing rate of neurons
Count Code	Count of number of neurons that spike during in time window
Binary Code	Binary pattern of length N from N neurons (e.g. 0010, where the third neuron spiked)
Temporal Code	Time of each spike in the spike train
Rank Code	Order in which neurons fire
Synchrony Code	Patterns from a population of neurons

Table 2.2: Coding Schemes in Spiking Neural Networks

2.2.2 Practical Benefits

Real-time Performance and Parallelism

The ability to communicate information in a small number of spikes has immense practical benefit.

First, the analog communication modeled in spiking neurons allow for naturally parallel and high-speed computation. The promise of real-time performance is a key motivator behind the development of neuromorphic hardware. Typical hardware such as the everyday computer uses the von Neumann architecture, which separates memory and processing, which has severe implications on performance. Neuromorphic hardware collocates memory and processing, overcoming the von Neumann bottleneck [10].

Power Efficiency

Next, the primary motivation cited in present-day literature is the power-efficiency of spiking neural networks. First, less energy is used in propagating fewer spikes. Second, communication via spikes is much more computationally efficient than the floating point arithmetic performed in artificial neural networks. This low energy footprint is a highly desirable trait in robotics applications.

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 [11]. Consider the chain rule in backpropagation:

$$\delta_j^\mu = g'\left(a_j^\mu\right) \sum_k w_{kj} \delta_k^\mu \tag{2.1}$$

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 feedback 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.

Online Learning and Fault Tolerance

Biological neurons brain evolve and adapt as time passes, creating new connections, and destroying old ones. While the phenomenon of *neuroplasticity* is not yet well understood, it is

hoped that the learning mechanisms behind human intelligence will inspire a new generation of online learning algorithms.

Neurons also frequently perish, and the human brain has the ability to self-heal. In the same way, spiking neural network architectures can be designed to possess the same self-healing and fault tolerance characteristics. This is a desirable trait for systems where uptime is critical.

In conclusion, the main motivations for spiking neural networks are the following:

- Neuromorphic hardware can overcome the von Neumann bottleneck
- When combined with neuromorphic hardware, SNNs can:
  - achieve real-time performance via asynchronous spiking communication
  - achieve low power consumption from sparse spiking computation
  - implement plasticity-inspired online learning mechanisms
  - Fault tolerance beyond what von Neumann architectures can provide
- Some SNNs have biologically-plausible neuron models, which can help answer open questions in neuroscience

2.3 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:

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

SRM describes the membrane potential of neuron i as:

$$u_i t = \eta(t - \hat{t_i}) + \sum_j \sum_f \epsilon_{ij}(t - t_j^{(f)}) + u_{\text{rest}} \tag{2.3}$$

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:

$$S_i(t) = \sum_f \delta\left(t - t_i^{(f)}\right) \tag{2.4}$$

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.

2.3.1 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 [12] and Intel's Loihi [3]. 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 [13], and solving graph problems by manual construction of the network graph [14].

2.4 Training Spiking Neural Networks

As explained in subsection 2.3.1, 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 section 2.2.2). 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 [15]:

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

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 [16] 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 [16], and using the reinforcement learning framework to train SNN [17, 18]. However, SNNs trained using the STDP learning rule have yet to achieve comparable performance compared to ANNs on relatively simple datasets like MNIST [11].

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 [19, 20]. 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 [21]. Others have modified the threshold function with a gate function [22], used the alpha transfer function to derive gradient update rules [9], and approximate the dirac-delta spikes with a probability density function [23].

Another approach is converting trained ANN models into SNNs [24]. 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 [25]. 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 [26].

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

Event-driven Visual-Tactile Sensing and Learning for Robots

In the introduction, we have alluded to VT-SNN, an event-driven visual-tactile perception system using spiking neural networks. This chapter describes the work done on that project. In section 3.1, we discuss the prior art for visual-tactile systems. In section 3.2, we discuss model architecture and model training. In section 3.3, we detail the robot hardware setup used across our experiments. Next, we discuss our experimental methods and results for our two robotics tasks: container classification (section 3.4), and slippage detection (section 3.5). We run the trained models on the Intel Loihi in section 3.6 to quantify the gains in power-efficiency and inference speeds. Finally, we conclude our work (section 3.7), and provide future research directions (section 3.8).

This work has been submitted to ROS 2020.

3.1 Related Work

In this section, we give an overview of related work on visual-tactile perception for robotics. Much of the work on event-based perception has been covered in chapter 2, and will not be repeated here.

3.1.1 Visual-Tactile Perception for Robots

3.2 Visual-Tactile Spiking Neural Network (VT-SNN)

3.3 Robot and Sensors Setup

3.4 Container & Weight Classification

3.5 Rotational Slippage Classification

3.6 Gains on Neuromorphic Hardware

3.7 Conclusion

3.8 Future work

Chapter 4

Reinforcement learning in Spiking Neural Networks

4.0.1 The Cartpole Environment

We use the Cartpole-v0 environment, a popular environment provided by OpenAI. The official description is as follows [27]:

> 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.

4.0.2 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

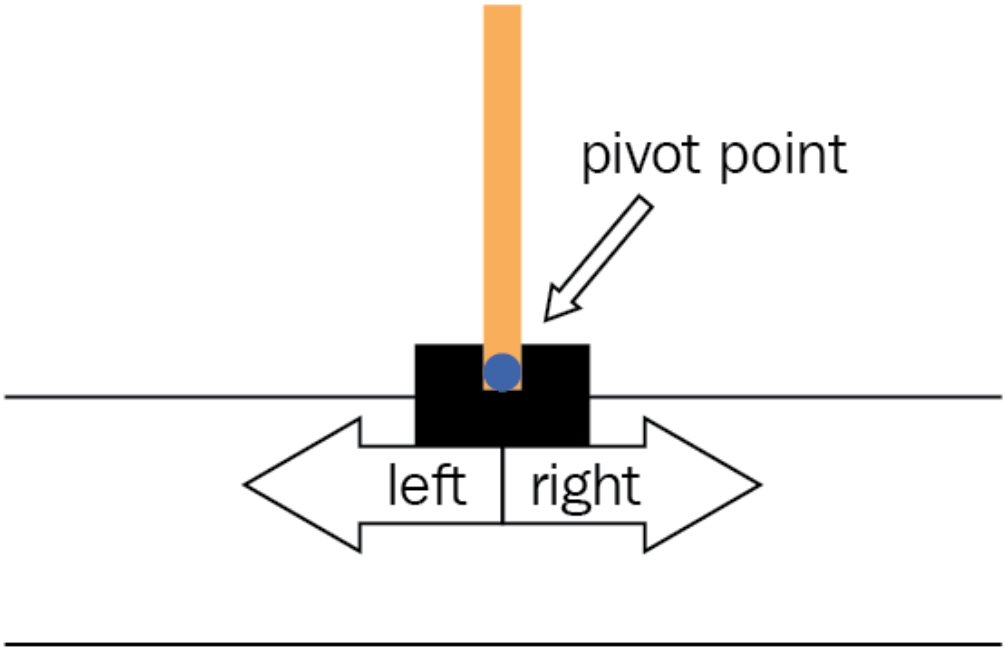

Figure 4.1: The OpenAI CartPole environment

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 [28] to log and ensure reproducibility of our experiments.[1]

Experiments show that the ANN model learns to solve the environment quickly when trained with VPG, as shown in Figure 4.2. The agent obtains an average episodic return of 200, "solving" the environment at 2000 time-steps.

4.0.3 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

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

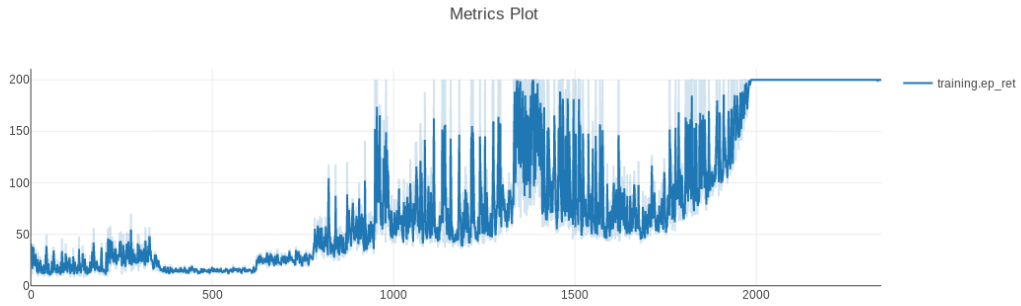

Figure 4.2: Plot of average episode returns over time-steps for a simple MLP agent.

cross-entropy [23].

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 4.3.

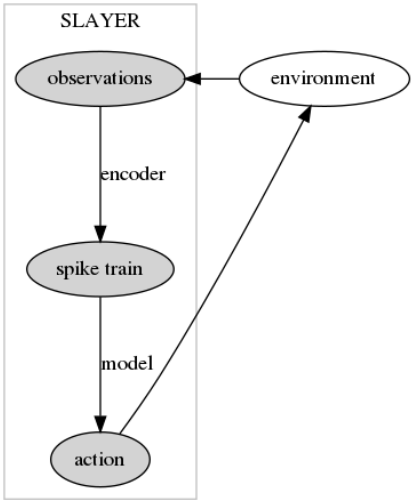

Figure 4.3: The SLAYER model requires an additional encoder to transform the observations into spike trains as input.

We encode the observations by modelling the observations as Poisson processes [29]. 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 4.4, the SLAYER model fails to learn to solve the problem.

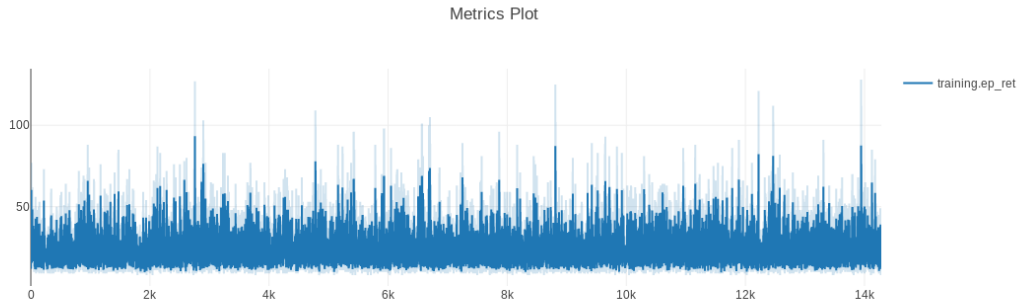

Figure 4.4: The SLAYER model with a Poisson encoder for the observations fails to learn to solve the problem.

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 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.

4.0.4 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 4.5. However, the agent is still able to learn from the

observations, suggesting that this environment is solvable by a SNN agent.

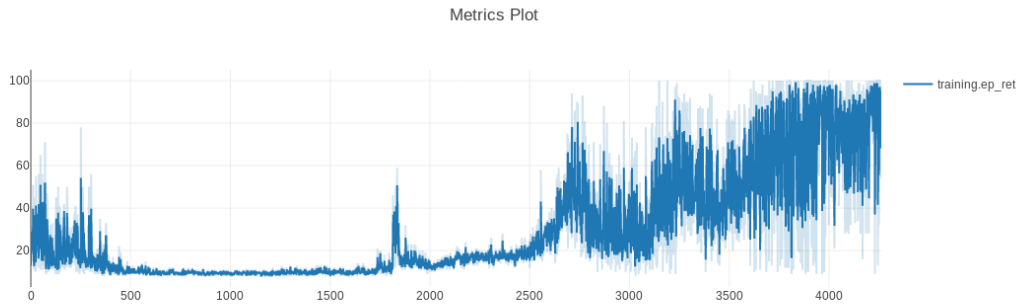

Figure 4.5: Plot of the average episode return of a CNN agent on the ImageCartpole environment over time.

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.

4.1 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.

Bibliography

[1] D. Li et al. “Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs”. In: *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*. Oct. 2016, pp. 477–484. DOI: 10.1109/BDCloud-SocialCom-SustainCom.2016.76.

[2] Michael Pfeiffer and Thomas Pfeil. “Deep learning with spiking neurons: opportunities and challenges”. In: *Frontiers in neuroscience* 12 (2018).

[3] Mike Davies et al. “Loihi: A neuromorphic manycore processor with on-chip learning”. In: *IEEE Micro* 38.1 (2018), pp. 82–99.

[4] Wang Wei Lee et al. “A neuro-inspired artificial peripheral nervous system for scalable electronic skins”. In: *Science Robotics* 4.32 (2019). DOI: 10.1126/scirobotics.aax2198. eprint: https://robotics.sciencemag.org/content/4/32/eaax2198.full.pdf. URL: https://robotics.sciencemag.org/content/4/32/eaax2198.

[5] Wolfgang Maass. “Networks of spiking neurons: The third generation of neural network models”. In: *Neural Networks* 10.9 (1997), pp. 1659–1671. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(97)00011-7. URL: http://www.sciencedirect.com/science/article/pii/S0893608097000117.

[6] Martin Stemmler. “A Single Spike Suffices: the Simplest Form of Stochastic Resonance in Model Neurons”. In: *Network: Computation in Neural Systems* 7.4 (1996), pp. 687–

716. DOI: 10.1088/0954-898x_7_4_005. URL: https://doi.org/10.1088/0954-898x_7_4_005.

[7] Simon Thorpe, Arnaud Delorme, and Rufin Van Rullen. “Spike-based strategies for rapid processing”. In: *Neural networks* 14.6-7 (2001), pp. 715–725.

[8] Robert Gütig. “To Spike, Or When To Spike?” In: *Current Opinion in Neurobiology* 25.nil (2014), pp. 134–139. DOI: 10.1016/j.conb.2014.01.004. URL: https://doi.org/10.1016/j.conb.2014.01.004.

[9] Iulia M. Comsa et al. “Temporal Coding in Spiking Neural Networks With Alpha Synaptic Function”. In: *CoRR* (2019). arXiv: 1907.13223 [cs.NE]. URL: http://arxiv.org/abs/1907.13223v2.

[10] John Backus. “Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs”. In: *Communications of the ACM* 21.8 (Aug. 1978), pp. 613–641. ISSN: 0001-0782. DOI: 10.1145/359576.359579. URL: http://dx.doi.org/10.1145/359576.359579.

[11] Amirhossein Tavanaei et al. “Deep learning in spiking neural networks”. In: *Neural Networks* 111 (2019), pp. 47–63. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2018.12.002. URL: http://www.sciencedirect.com/science/article/pii/S0893608018303332.

[12] Paul A. Merolla et al. “A million spiking-neuron integrated circuit with a scalable communication network and interface”. In: *Science* 345.6197 (2014), pp. 668–673. ISSN: 0036-8075. DOI: 10.1126/science.1254642. eprint: https://science.sciencemag.org/content/345/6197/668.full.pdf. URL: https://science.sciencemag.org/content/345/6197/668.

[13] Guangzhi Tang, Arpit Shah, and Konstantinos P. Michmizos. “Spiking Neural Network on Neuromorphic Hardware for Energy-Efficient Unidimensional Slam”. In: *CoRR* (2019). arXiv: 1903.02504 [cs.RO]. URL: http://arxiv.org/abs/1903.02504v2.

[14] William Severa et al. “Spiking network algorithms for scientific computing”. In: *2016 IEEE International Conference on Rebooting Computing (ICRC)* (2016), pp. 1–8.

[15] Alexander Sboev et al. “Spiking Neural Network Reinforcement Learning Method Based on Temporal Coding and Stdp”. In: *Procedia Computer Science* 145.nil (2018), pp. 458–463. DOI: 10.1016/j.procs.2018.11.107. URL: https://doi.org/10.1016/j.procs.2018.11.107.

[16] Hananel Hazan et al. “BindsNET: A Machine Learning-Oriented Spiking Neural Networks Library in Python”. In: *Frontiers in Neuroinformatics* 12 (2018), p. 89. ISSN: 1662-5196. DOI: 10.3389/fninf.2018.00089. URL: https://www.frontiersin.org/article/10.3389/fninf.2018.00089.

[17] Zhenshan Bing et al. “Supervised Learning in SNN via Reward-Modulated Spike-Timing-Dependent Plasticity for a Target Reaching Vehicle”. In: *Frontiers in Neurorobotics* 13 (2019), p. 18. ISSN: 1662-5218. DOI: 10.3389/fnbot.2019.00018. URL: https://www.frontiersin.org/article/10.3389/fnbot.2019.00018.

[18] Chankyu Lee et al. “Training Deep Spiking Convolutional Neural Networks With STDP-Based Unsupervised Pre-training Followed by Supervised Fine-Tuning”. In: *Frontiers in Neuroscience* 12 (2018), p. 435. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00435. URL: https://www.frontiersin.org/article/10.3389/fnins.2018.00435.

[19] Robert Urbanczik and Walter Senn. “A Gradient Learning Rule for the Tempotron”. In: *Neural Computation* 21.2 (2009), pp. 340–352. DOI: 10.1162/neco.2008.09-07-605. URL: https://doi.org/10.1162/neco.2008.09-07-605.

[20] Jun Lee, Tobi Delbruck, and Michael Pfeiffer. “Training Deep Spiking Neural Networks Using Backpropagation”. In: *Frontiers in Neuroscience* 10 (Aug. 2016). DOI: 10.3389/fnins.2016.00508.

[21] Sander Bohte, Joost Kok, and Johannes Poutré. “SpikeProp: backpropagation for networks of spiking neurons.” In: vol. 48. Jan. 2000, pp. 419–424.

[22] Dongsung Huh and Terrence J Sejnowski. “Gradient Descent for Spiking Neural Networks”. In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 1433–1443. URL: http://papers.nips.cc/paper/7417-gradient-descent-for-spiking-neural-networks.pdf.

[23] Sumit Bam Shrestha and Garrick Orchard. “SLAYER: Spike Layer Error Reassignment in Time”. In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 1412–1421. URL: http://papers.nips.cc/paper/7415-slayer-spike-layer-error-reassignment-in-time.pdf.

[24] Bodo Rueckauer et al. “Theory and Tools for the Conversion of Analog To Spiking Convolutional Neural Networks”. In: *CoRR* (2016). arXiv: 1612.04052 [stat.ML]. URL: http://arxiv.org/abs/1612.04052v1.

[25] Benjamin Scellier and Yoshua Bengio. “Equilibrium Propagation: Bridging the Gap between Energy-Based Models and Backpropagation”. In: *Frontiers in Computational Neuroscience* 11 (2017), p. 24. ISSN: 1662-5188. DOI: 10.3389/fncom.2017.00024. URL: https://www.frontiersin.org/article/10.3389/fncom.2017.00024.

[26] Peter O’Connor, Efstratios Gavves, and Max Welling. “Training a Spiking Neural Network with Equilibrium Propagation”. In: *Proceedings of Machine Learning Research*. Ed. by Kamalika Chaudhuri and Masashi Sugiyama. Vol. 89. Proceedings of Machine Learning Research. PMLR, 16–18 Apr 2019, pp. 1516–1523. URL: http://proceedings.mlr.press/v89/o-connor19a.html.

[27] OpenAI. *OpenAI Gym*. https://gym.openai.com/envs/CartPole-v0/. Online; accessed 02 November 2019. 2019.

[28] Klaus Greff et al. “The Sacred Infrastructure for Computational Research”. In: *Proceedings of the 16th Python in Science Conference*. Ed. by Katy Huff et al. 2017, pp. 49–56. DOI: 10.25080/shinma-7f4c6e7-008.

[29] David Heeger. “Poisson model of spike generation”. In: *Handout, University of Standford* 5 (2000), pp. 1–13.