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

DRAFT! — March 23, 2020 — DRAFT!

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

List of Figures

2.1 The biological neuron [13], in contrast with a perceptron. 10
2.2 A taxonomy of popular spiking neuron models. 11

4.1 The OpenAI CartPole environment 24
4.2 Plot of average episode returns over time-steps for a simple MLP agent. 25
4.3 The SLAYER model requires an additional encoder to transform the observations into spike trains as input. 25
4.4 The SLAYER model with a Poisson encoder for the observations fails to learn to solve the problem. 26
4.5 Plot of the average episode return of a CNN agent on the ImageCartpole environment over time. 27

List of Tables

2.1 The three generations of neural networks. 5
2.2 Coding Schemes in Spiking Neural Networks 7
2.3 A taxonomy of learning algorithms for spiking neuron models. 15

Contents

Title i

Abstract ii

List of Figures iii

List of Tables iv

1 Introduction 1
- 1.1 Research Outline . . . 2
- 1.2 Individual Contributions . . . 3

2 Background 4
- 2.1 The Generations of Neural Networks . . . 4
- 2.2 Motivating Spiking Neural Networks . . . 5
  - 2.2.1 Information Encoding via Spikes . . . 5
  - 2.2.2 Practical Benefits . . . 7
  - 2.2.3 Biological Plausibility . . . 8
  - 2.2.4 Summary . . . 8
- 2.3 Neuron Models . . . 9
  - 2.3.1 McCulloch-Pitts . . . 10
  - 2.3.2 Biologically-plausible Models . . . 10
  - 2.3.3 Biologically-inspired Models . . . 10
- 2.4 The Spike-response Model . . . 11
- 2.5 Neuromorphic Hardware . . . 13
- 2.6 Training Spiking Neural Networks . . . 14
  - 2.6.1 Gradient-based methods . . . 17
  - 2.6.2 Conclusion . . . 19
  - 2.6.3 Future Work . . . 19

3 Event-driven Visual-Tactile Sensing and Learning for Robots 21
- 3.1 Related Work . . . 21
  - 3.1.1 Visual-Tactile Perception for Robots . . . 22
- 3.2 Visual-Tactile Spiking Neural Network (VT-SNN) . . . 22
- 3.3 Robot and Sensors Setup . . . 22
- 3.4 Container & Weight Classification . . . 22
- 3.5 Rotational Slippage Classification . . . 22
- 3.6 Gains on Neuromorphic Hardware . . . 22
- 3.7 Conclusion . . . 22
- 3.8 Future work . . . 22

4 Reinforcement learning in Spiking Neural Networks **23**
4.0.1 The Cartpole Environment 23
4.0.2 Training a Simple ANN agent 24
4.0.3 Training a SNN agent 25
4.0.4 The ImageCartpole Environment 26
4.1 Future Work 27

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 ()FIXME: fill this in 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 (2.6), and give a short conclusion with 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 that use a memory mechanism, such as the LSTM [10], require 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. 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

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

Real-time Performance and Parallelism

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 [11].

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.

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 *neuroplasticticity* 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.

2.2.3 Biological Plausibility

In addition to having practical benefits, spiking neural networks more closely model the biological brain. 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 difficult to reconcile.

First, neurons in ANNs communicate via continuous-valued activations. In contrast, it is well established that 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 [12], explained in further detail in section 2.6. This result motivates the search for alternative learning mechanisms for spiking neural networks.

2.2.4 Summary

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 Neuron Models

Neuron models are mathematical models that describe the underlying neuron dynamics: how electrical signals propagate from inputs to outputs, and the changes in the neurons internal state, such as the membrane potential.

Many of these neuron models are modeled after the biological neuron, which has a few main components:

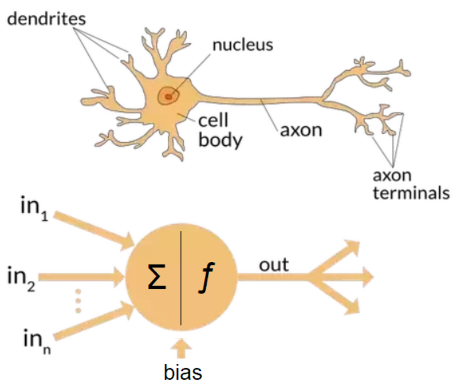
Dendrites Neurons receive input signals at the dendrites, which transmit incoming electrical signals to the cell body.

Cell Body The cell body is responsible for "processing" the input from the dendrites. The typical behaviour is to accumulate the electrical input from the dendrites. Once the cell's membrane potential crosses a certain threshold, it spikes, and sends an output signal through the axon.

Axons The axon receives electrical signals from the cell body. Electrical signals propagate down to axon terminals, which control the release of neurotransmitters.

The perceptron was biologically-inspired, and has an analogous representation that may be helpful for understanding, which we present in Figure 2.1.

In this section, we present a taxonomy of neuron models. These neuron models vary in computational and implementation complexity. We discuss some of them in brief, and conduct a thorough review of the Spike Response Model (SRM). For a more in-depth treatment of spiking neuron models, we refer you to [14].



Biological Neuron	Perceptron
Dendrites	Inputs
Cell Body	Processing input: $f(\sum \mathbf{w} \cdot \mathbf{i})$
Axon (Terminals)	Output

Figure 2.1: The biological neuron [13], in contrast with a perceptron.

2.3.1 McCulloch-Pitts

Activation functions such as the sigmoid function, and hyperbolic tangent function can be implemented at the hardware level in the McCulloch-Pitts neuron model. However, these models use digital input and output, which means they have a higher footprint than spiking neuron models which perform analog computation.

2.3.2 Biologically-plausible Models

The Hodgkin-Huxley [15] model is a mathematical model that uses four-dimensional non-linear differential equations to model the neuron dynamics, using the transfer of ions as the medium. This model is popular among the neuroscientific community, for accurately modelling the biological process. The Morris-Lecar [16] model is a simplification of the Hodgkin-Huxley model, which is more easily implementable in hardware.

2.3.3 Biologically-inspired Models

Some biologically-inspired models often sacrifice biological-plausibility for ease of implementation. These models have a smaller number of parameters, resulting in models that are not only easier to train, but also have a smaller energy footprint. These models include the FitzHugh-Nagumo [17], and the Izhikevich [18] model.

A particularly notable family of biologically-inspired models in the integrate-and-fire family.

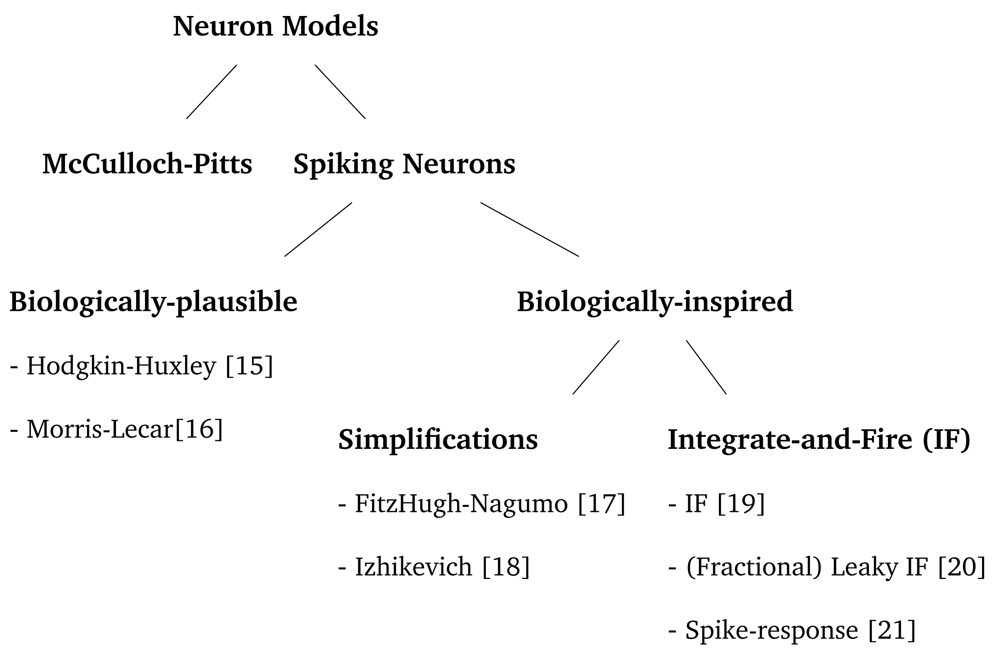


Figure 2.2: A taxonomy of popular spiking neuron models.

These models maintain as state, the membrane potential of the neuron, accumulating charge from input neurons. The leaky integrate-and-fire model introduces an additional "leak" term, which controls the decay of the neuron's membrane potential over time.

2.4 The Spike-response Model

The spike-response model [21] is a generalization of the leaky integrate-and-fire model. We review this model because this neuron model is used in SLAYER [22], the framework we use in our experiments.

Rather than modelling the neuronal dynamics using differential equations, the SRM model uses convolutional filters, a property that makes the simulation and training of these spiking models easy. In contrast to the leaky IF model, the refractory behaviour of the neuron is also captured, which has allowed it to better fit spiking data [23].

In this model, a neuron is characterized by a single variable μ, the membrane potential of the cell. Suppose that the neuron has fired its last spike at time $\hat{t}$. Then, in the most general formulation, at each time $t > \hat{t}$, the state of the cell is expressed as an integral over the previous timesteps:

$$\mu(t) = \eta(t-\hat{t}) + \int_{-\infty}^{+\infty} \kappa(t-\hat{t}, s) I(t-s) ds \tag{2.1}$$

where η describes the form of the action potential, κ is the linear response to an input pulse, and $I(t)$ is the stimulating current. In most spiking neural network architectures, the imposed current $I(t)$ is instead described as a weighted sum of presynaptic inputs, and the equation can be rewritten as:

$$\mu(t) = \eta(t-\hat{t}) + \sum_j \sum_f w_j \epsilon(t-\hat{t}, t-t_j^f) \tag{2.2}$$

where ϵ, the spike response kernel, is obtained by convolving the kernel κ with the time course of the presynaptic input.

A common form for the refractory kernel η is:

$$\eta(t) = -\eta_0 \text{exp}\left(-\frac{t}{\tau}\right) \mathcal{H}(t) \tag{2.3}$$

where η_0 is the amplitude of the relative refractorines, τ is a decay time constant, and $\mathcal{H}$ is the Heaviside step function.

SLAYER uses the SRM model, and formulates it this way. Suppose that the neuron receives an input spike train, $s_i(t) = \sum_f \delta\left(t - t_i^{(f)}\right)$. The spike train is represented as a sum of Dirac delta functions with $\delta(x) = 0$ for $x \neq 0$ and $\int_{-\infty}^{\infty} \delta(x) dx = 1$. In the spike response model, incoming spikes are converted into a *spike response signal*, $a_i(t)$, by convolving $s_i(t)$ with a spike response kernel $\epsilon(\cdot)$: $a_i(t) = (\epsilon * s_i)(t)$. The refractory response of the neuron can also be represented with a refractory kernel $\eta(\cdot)$, giving the formula for the refractory response: $(\eta * s)(t)$ for the neuron's spike train $s(t)$.

Each spike response signal is scaled by a synaptic weight w_i, to generate a post synaptic potential (PSP). The neuron's state is then given by the equation:

$$\mu(t) = (\eta * s)(t) + \sum_i w_i(\epsilon * s_i)(t) = (\eta * s)(t) + \boldsymbol{w}^T \boldsymbol{a}(t) \tag{2.4}$$

The neuron produces an output spike when membrane threshold θ is reached. The spike function $f_s(\cdot)$ is defined as:

$$f_s(\mu) : \mu \to s, s(t) := s(t) + \delta(t - t^{f+1}) \text{ where } t^{f+1} = \min \left\{ t : \mu(t) = \theta, t > t^f \right\} \tag{2.5}$$

2.5 Neuromorphic Hardware

To obtain the full benefits of spiking neural networks, they need to be deployed on specialized neuromorphic hardware. In a traditional Von Neumann architecture, the logic core operates on data fetched sequentially from memory. In neuromorphic chips, both computation and memory are instead distributed across computational units that are connected via synapses. Henc, the neuronal architecture and parameters 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.

Examples of such chips include IBM's TrueNorth [24] and Intel's Loihi [3]. Loihi features 128 neuromorphic cores, each of which implementing 1024 spiking neural units. The mesh protocol further supports scaling up to 4096 on-chip cores. The large number of computational units and their interconnectivity is a common feature in neuromorphic chips.

Blouw, Choo, Hunsberger, *et al.* quantified the trade-offs between network size, inference speed, and energy cost between different hardware. The comparison showed that the Intel Loihi was 100 times more efficient than a GPU, with inference speeds also being three times faster [25].

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 [26], and solving graph problems by manual construction of the network graph [27].

2.6 Training Spiking Neural Networks

It is desirable to train spiking neural networks to perform arbitrary tasks, utilizing power-efficient neuromorphic chips that break the Von Neumann bottleneck. However, spiking neural networks have yet to achieve superior performance relative to their ANN counterparts. In this section, we first discuss the difficulties in training SNNs. Then, we provide a taxonomy of training strategies.

Difficulties in Training

Because backpropagation has been effective in training artificial neural networks, the natural approach is to attempt to apply it to spiking neural networks. However, consider the chain rule in backpropagation:

$$\delta_j^\mu = g'\left(a_j^\mu\right) \sum_k w_{kj} \delta_k^\mu \tag{2.6}$$

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 key challenges. 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. The non-differentiability.

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. This is also known as the “weight-transport” problem [28]. A more practical problem the usage of non-local information during error assignment. This makes the backpropagation incompatible with many neuromorphic hardware implementations, such as the Loihi: the learning rules of the Loihi are constrained to be of a sum-of-products form.

In the following sections, we present training methods that work around these difficulties, according to the following classification:

Usage of Gradients Recent advancements in the field have opened up the possibilities of using gradient-based methods for training spiking neural networks. We classify the learning algorithms based on whether they utilize gradient information for learning.

Offline/Online Since many neuromorphic hardware implementations impose strict restrictions on the neuron update rules, many learning algorithms use simulations on Von Neumann hardware to learn the SNN weights before deploying them on hardware. We classify the learning algorithms based on whether online learning on neuromorphic chips is possible.

	Non-gradient based	**Gradient-based**
Offline-only	Evolutionary Algorithms	ANN $\rightarrow$ SNN Smooth models Surrogate Gradients
Online	Local Learning Rules	Surrogate Gradients + Random BP

Table 2.3: A taxonomy of learning algorithms for spiking neuron models.

Local Learning Rules

Local learning rules are simple fixed rules, that tells a neuron how to update its weights for each training example. These rules often only use information from neighbouring neurons, and can be easily implemented on neuromorphic hardware. The earliest of such rules is Hebbian learning [29], which can be summarized as “neurons which fire together wire together”.

A recent, and much more popular learning rule is the spike-timing-dependent plasticity rule (STDP). 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.

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 [30]:

$$\Delta w = \begin{cases} -\alpha\lambda \cdot \exp\left(-\frac{t_{\text{pre}}-t_{\text{post}}}{\tau_-}\right), \text{if } t_{\text{pre}} - t_{\text{post}} > 0 \\ \lambda \cdot \exp\left(-\frac{t_{\text{post}}-t_{\text{pre}}}{\tau_+}\right), \text{if } t_{\text{pre}} - t_{\text{post}} < 0 \end{cases} \tag{2.7}$$

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

Because the rules are local, these local learning rules fall within the unsupervised learning paradigm of machine learning. However, more recently, variants of STDP such as reward-modulated STDP [31] have been proposed to allow for neurons to learn to update synaptic weights to maximize the global reward of a system.

Libraries such as BindsNET [32] that simulate SNNs on Von Neumann computers implementing these rules. Recent attempts have been made to combine Reinforcement Learning and STDP: both in solving reinforcement learning problems [32], and using the reinforcement learning framework to train SNN [33], [34]. However, SNNs trained using the STDP learning rule have yet to achieve comparable performance compared to ANNs on relatively simple datasets like MNIST [12].

Evolutionary Algorithms

In the absence of gradients, neuroevolution is a popular option. Evolutionary algorithms were once thought to be ineffective for training neural networks, which have large parameter spaces. However, advancements in the algorithmic front have made them feasible for solving challenging reinforcement learning problems [35].

One such evolutionary algorithm that can be easily applied to spiking neural networks is differential evolution (DE). In differential evolution, a population of candidate solutions are maintained. At each timestep, the parameters of these agents are tweaked, and the candidates is evaluated. In some variants of DE, a reproduction step involving crossing-over of weights between candidate solutions is also introduced. The agent that has the best performance is returned after some stopping criterion (e.g. time-step) is reached. Evolutionary algorithms tend to be computationally expensive, but parallelizable. For example, in DE the evaluation of candidates can be done in parallel. Hence, it requires offline training. DE has been successfully applied to achieve comparable results with ANNs on simple datasets like IRIS [36].

2.6.1 Gradient-based methods

The class of gradient-based methods use several techniques to overcome the difficulties in training presented in section 2.6.

Converting ANNs into SNNs

One approach is converting trained ANN models into SNNs [37], [38]. One can achieve the same input-output mapping as a deep ANN with a converted SNN. Common ANN layers such as softmax, batch normalization and max-pooling layers have their corresponding spiking counterparts. This method allows for using the full toolkit of deep-learning methods in training the ANNs prior to conversion. However, only a subset of ANN architectures can be converted into SNNs. For example, the activations need to be always non-negative, to match the firing rates of SNNs. These converted ANNs are also limited to using rate-codes, which as discussed in section 2.2 is an inefficient spike encoding.

Smooth models

All gradient-based methods applied directly to SNNs require overcoming the discontinuous, binary nature of spike trains.One simple solution is to use smooth models.

First, neuron models with a smooth spike generating process like the Hodgkin-Huxley, Morris-Lecar and FitzHugh-Nagumo models need no further approximation techniques. Next, stochastic models also allow the definition of gradients on expectations, allowing the loglikelihood of a spike train to be optimized using gradient descent [39].

Surrogate Gradients

Rather than using a smooth model, surrogate gradient methods allow for backpropagation without using a smooth model. Surrogate gradient methods replace the spiking non-linearity by the derivative of a smooth function. 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 [40]. Others have modified the threshold function with a gate function [41], used the alpha transfer function to derive gradient update rules [9], and approximate the Dirac delta spikes with a probability density function [22].

Online Gradient-based Learning Rules

While surrogate gradient methods enable backpropagation, these methods use non-local information and cannot be implemented on neuromorphic hardware. 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 [42], [43].

One can relax the requirements of gradient backpropagation [44], to achieve local learning rules. Random backpropagation algorithms sidestep the problem of non-locality, by replacing the weights in the backpropagation rule with random weights. On some benchmark tasks, this lead to little loss in performance. SuperSpike uses random feedback weights, and surrogate gradients to derive a biologically-plausible, online learning rule suitable for hardware implementations [45]. DECOLLE also uses surrogate gradients in a linear IF neuron model, using a local and rate-based cost function to derive local update rules from partial first principles [46].

Equilibrium Propagation was recently proposed to solve the neurobiological incompatibilities of backpropagation [47]. 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 [48].

2.6.2 Conclusion

In this chapter, we have discussed the principles and motivations behind spiking neural networks. We also presented a taxonomy of different spiking neuron models, and their tradeoffs. We focused in particular on the integrate-and-fire family, specifically the spike response model, because of its widespread adoption in the neuromorphic community. We then discussed the challenges in training spiking neuron models, and techniques to side-step them.

The gold standard is an online learning rule that uses only local information, that can be implemented on neuromorphic hardware. We have seen methods such as SuperSpike and DECOLLE, which combine a variety of approximation techniques to derive gradient-based local update rules. These methods do not yet scale well to deep SNNs. How to perform credit assignment using local information in deep networks is still an open research question.

2.6.3 Future Work

In this review, we have spoken mainly about the techniques and the theory, but have little mention of the experiments performed. There are unsurprisingly few publicly available event-based datasets. Many research papers take an existing non-spiking dataset such as MNIST, and convert them into spike trains using a Poisson-encoding process. Others use the N-MNIST and N-Caltech101 dataset [49], or the DvsGesture [50] dataset. The spike trains in these dataset are much less sparse than the naturally occurring spikes in biological neurons. In addition, both SNNs and ANNs demonstrate strong experimental results on these datasets, which may suggest that they are too easy. We would like to see more diverse and challenging publicly-available spiking datasets, and strong and reliable benchmarking for the various SNN architectures and learning methods.

There has also been a resurgence of interest in combining differential equation solvers with neural networks [51]. All the learning methods described in this chapter require the discretization in time. Since the neuronal dynamics of many spiking neurons can be represented as differential equations, continuous-time SNNs may be worth looking into.

Finally, a future direction of research may be the interpretation of spiking neural networks. In deep learning, we have come to learn that convolutional neural networks learn low-level features in lower layers, and higher level features in deeper layers by visualizing the learned filters. Understanding what the spiking neural network is learning at each layer, and how the different layers interact may provide answers to open neuroscience questions, as well as open up new research directions in learning algorithms.

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

FIXME: Introduction FIXME: RL survey

4.0.1 The Cartpole Environment

We use the Cartpole-v0 environment, a popular environment provided by OpenAI. The official description is as follows [52]:

> 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.

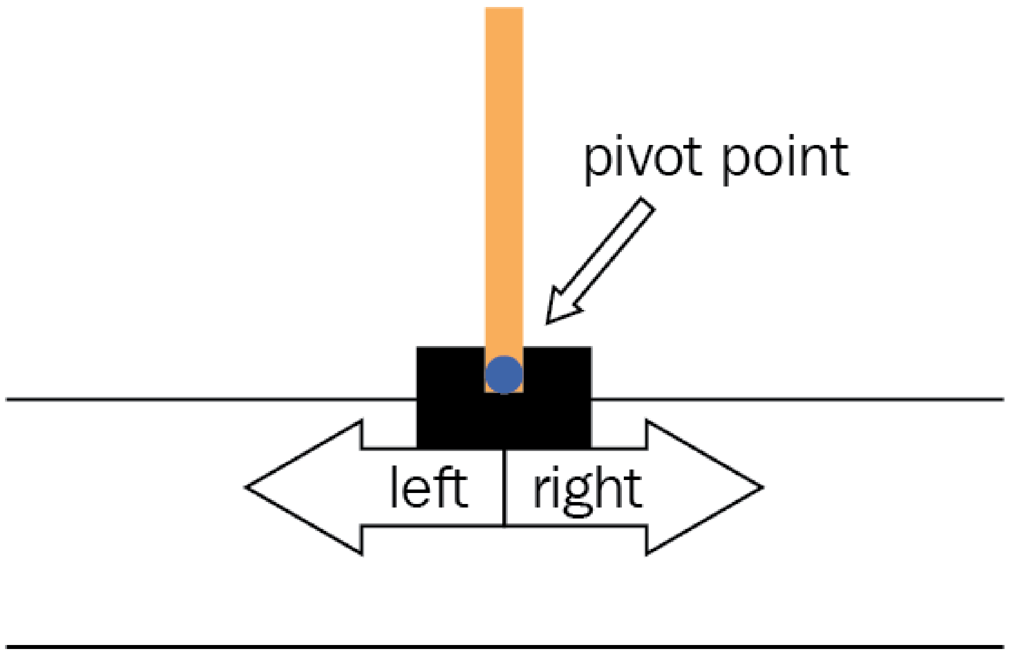


Figure 4.1: The OpenAI CartPole environment

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 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 [53] 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.

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

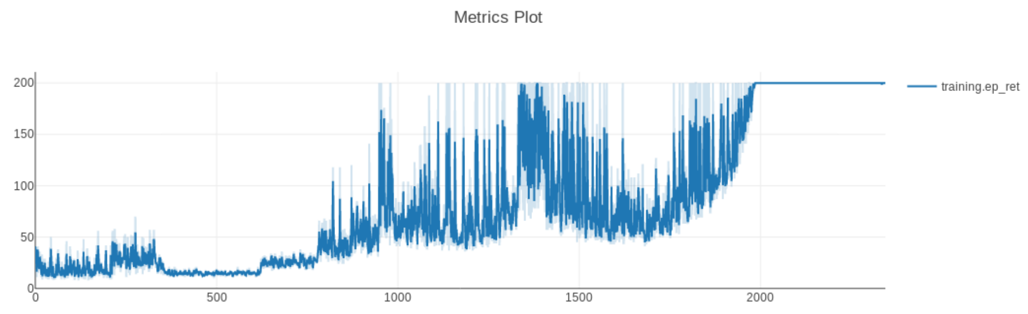


Figure 4.2: Plot of average episode returns over time-steps for a simple MLP agent.

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 cross-entropy [22].

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.

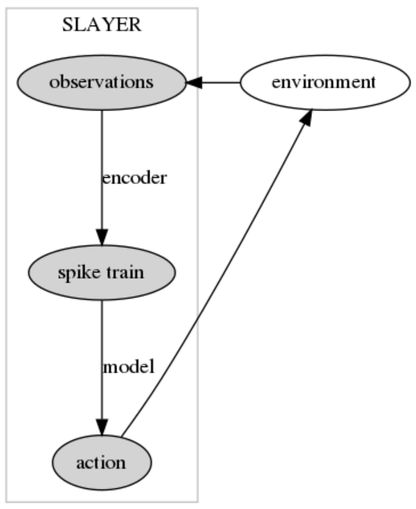


Figure 4.3: The SLAYER model requires an additional encoder to transform the observations into spike trains as input.

We encode the observations by modelling the observations as Poisson processes [54]. 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.

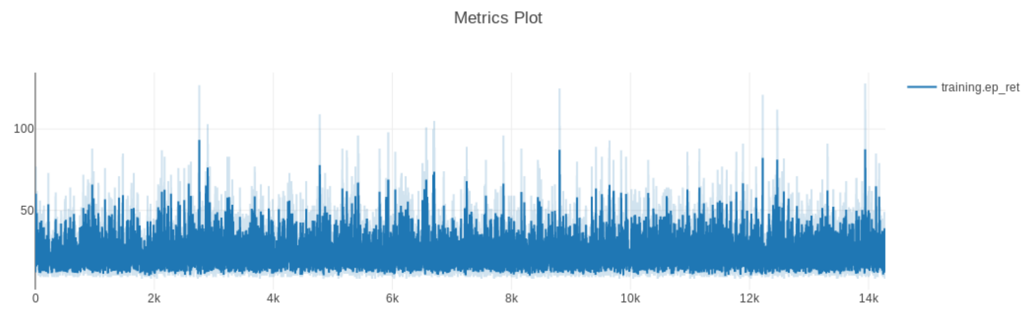


Figure 4.4: The SLAYER model with a Poisson encoder for the observations fails to learn to solve the problem.

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.

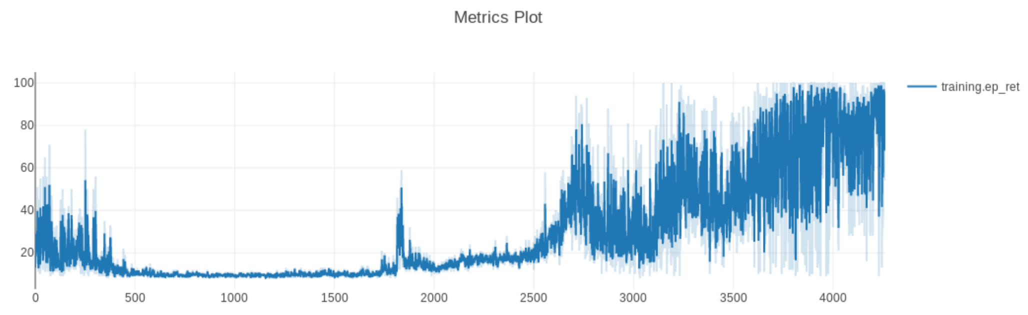


Figure 4.5: Plot of the average episode return of a CNN agent on the ImageCartpole environment over time.

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, X. Chen, M. Becchi, and Z. Zong, “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.

[2] M. Pfeiffer and T. Pfeil, “Deep learning with spiking neurons: Opportunities and challenges,” *Frontiers in neuroscience*, vol. 12, 2018.

[3] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, *et al.*, “Loihi: A neuromorphic manycore processor with on-chip learning,” *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[4] W. W. Lee, Y. J. Tan, H. Yao, S. Li, H. H. See, M. Hon, K. A. Ng, B. Xiong, J. S. Ho, and B. C. K. Tee, “A neuro-inspired artificial peripheral nervous system for scalable electronic skins,” *Science Robotics*, vol. 4, no. 32, 2019. eprint: `https://robotics.sciencemag.org/content/4/32/eaax2198.full.pdf`. [Online]. Available: `https://robotics.sciencemag.org/content/4/32/eaax2198`.

[5] W. Maass, “Networks of spiking neurons: The third generation of neural network models,” *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997, ISSN: 0893-6080. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0893608097000117`.

[6] M. Stemmler, “A single spike suffices: The simplest form of stochastic resonance in model neurons,” *Network: Computation in Neural Systems*, vol. 7, no. 4, pp. 687–716, 1996. [Online]. Available: `https://doi.org/10.1088/0954-898x_7_4_005`.

[7] S. Thorpe, A. Delorme, and R. Van Rullen, “Spike-based strategies for rapid processing,” *Neural networks*, vol. 14, no. 6-7, pp. 715–725, 2001.

[8] R. Gütig, “To spike, or when to spike?” *Current Opinion in Neurobiology*, vol. 25, no. nil, pp. 134–139, 2014. [Online]. Available: `https://doi.org/10.1016/j.conb.2014.01.004`.

[9] I. M. Comsa, K. Potempa, L. Versari, T. Fischbacher, A. Gesmundo, and J. Alakuijala, “Temporal coding in spiking neural networks with alpha synaptic function,” *CoRR*, 2019. arXiv: `1907.13223 [cs.NE]`. [Online]. Available: `http://arxiv.org/abs/1907.13223v2`.

[10] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] J. Backus, “Can programming be liberated from the von neumann style?: A functional style and its algebra of programs,” *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, Aug. 1978, ISSN: 0001-0782. [Online]. Available: `http://dx.doi.org/10.1145/359576.359579`.

[12] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, “Deep learning in spiking neural networks,” *Neural Networks*, vol. 111, pp. 47–63, 2019, ISSN: 0893-6080. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0893608018303332`.

[13] R. Nagyfi. (2018). “The differences between artificial and biological neural networks,” [Online]. Available: `https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7`.

[14] E. M. Izhikevich, “Which model to use for cortical spiking neurons?” *IEEE transactions on neural networks*, vol. 15, no. 5, pp. 1063–1070, 2004.

[15] A. L. Hodgkin and A. F. Huxley, “A quantitative description of membrane current and its application to conduction and excitation in nerve,” *The Journal of physiology*, vol. 117, no. 4, pp. 500–544, 1952.

[16] C. Morris and H. Lecar, “Voltage oscillations in the barnacle giant muscle fiber,” *Biophysical Journal*, vol. 35, no. 1, pp. 193–213, 1981. [Online]. Available: `https://doi.org/10.1016/s0006-3495(81)84782-0`.

[17] R. FitzHugh, “Mathematical models of threshold phenomena in the nerve membrane,” *The bulletin of mathematical biophysics*, vol. 17, no. 4, pp. 257–278, 1955.

[18] E. M. Izhikevich, “Simple model of spiking neurons,” *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[19] L. Abbott, “Lapicque’s introduction of the integrate-and-fire model neuron (1907),” *Brain Research Bulletin*, vol. 50, no. 5-6, pp. 303–304, 1999. [Online]. Available: `https://doi.org/10.1016/s0361-9230(99)00161-6`.

[20] W. Teka, T. M. Marinov, and F. Santamaria, “Neuronal spike timing adaptation described with a fractional leaky integrate-and-fire model,” *PLoS Computational Biology*, vol. 10, no. 3, e1003526, 2014. [Online]. Available: `https://doi.org/10.1371/journal.pcbi.1003526`.

[21] W. Gerstner, “A framework for spiking neuron models-the spike response model,” *Handbook of Biological Physics*, vol. 4, no. BOOK_CHAP, pp. 469–516, 2001.

[22] S. B. Shrestha and G. Orchard, “Slayer: Spike layer error reassignment in time,” in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., Curran Associates, Inc., 2018, pp. 1412–1421. [Online]. Available: `http://papers.nips.cc/paper/7415-slayer-spike-layer-error-reassignment-in-time.pdf`.

[23] R. Jolivet, T. J. Lewis, and W. Gerstner, “Generalized integrate-and-fire models of neuronal activity approximate spike trains of a detailed model to a high degree of accu-

racy," *Journal of Neurophysiology*, vol. 92, no. 2, pp. 959–976, 2004. [Online]. Available: `https://doi.org/10.1152/jn.00190.2004`.

[24] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014, ISSN: 0036-8075. eprint: `https://science.sciencemag.org/content/345/6197/668.full.pdf`. [Online]. Available: `https://science.sciencemag.org/content/345/6197/668`.

[25] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith, "Benchmarking keyword spotting efficiency on neuromorphic hardware," *CoRR*, 2018. arXiv: `1812.01739 [cs.LG]`. [Online]. Available: `http://arxiv.org/abs/1812.01739v2`.

[26] G. Tang, A. Shah, and K. P. Michmizos, "Spiking neural network on neuromorphic hardware for energy-efficient unidimensional slam," *CoRR*, 2019. arXiv: `1903.02504 [cs.RO]`. [Online]. Available: `http://arxiv.org/abs/1903.02504v2`.

[27] W. Severa, O. Parekh, K. D. Carlson, C. D. James, and J. B. Aimone, "Spiking network algorithms for scientific computing," *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–8, 2016.

[28] S. Grossberg, "Competitive learning: From interactive activation to adaptive resonance," *Cognitive science*, vol. 11, no. 1, pp. 23–63, 1987.

[29] D. O. Hebb, *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall, 1949.

[30] A. Sboev, D. Vlasov, R. Rybka, and A. Serenko, "Spiking neural network reinforcement learning method based on temporal coding and stdp," *Procedia Computer Science*, vol. 145, no. nil, pp. 458–463, 2018. [Online]. Available: `https://doi.org/10.1016/j.procs.2018.11.107`.

[31] R. Legenstein, D. Pecevski, and W. Maass, “A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback,” *PLoS Computational Biology*, vol. 4, no. 10, e1000180, 2008. [Online]. Available: `https://doi.org/10.1371/journal.pcbi.1000180`.

[32] H. Hazan, D. J. Saunders, H. Khan, D. Patel, D. T. Sanghavi, H. T. Siegelmann, and R. Kozma, “Bindsnet: A machine learning-oriented spiking neural networks library in python,” *Frontiers in Neuroinformatics*, vol. 12, p. 89, 2018, ISSN: 1662-5196. [Online]. Available: `https://www.frontiersin.org/article/10.3389/fninf.2018.00089`.

[33] Z. Bing, I. Baumann, Z. Jiang, K. Huang, C. Cai, and A. Knoll, “Supervised learning in snn via reward-modulated spike-timing-dependent plasticity for a target reaching vehicle,” *Frontiers in Neurorobotics*, vol. 13, p. 18, 2019, ISSN: 1662-5218. [Online]. Available: `https://www.frontiersin.org/article/10.3389/fnbot.2019.00018`.

[34] C. Lee, P. Panda, G. Srinivasan, and K. Roy, “Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning,” *Frontiers in Neuroscience*, vol. 12, p. 435, 2018, ISSN: 1662-453X. [Online]. Available: `https://www.frontiersin.org/article/10.3389/fnins.2018.00435`.

[35] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, “Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning,” *CoRR*, 2017. arXiv: `1712.06567 [cs.NE]`. [Online]. Available: `http://arxiv.org/abs/1712.06567v3`.

[36] N. Pavlidis, O. Tasoulis, V. Plagianakos, G. Nikiforidis, and M. Vrahatis, “Spiking neural network training using evolutionary algorithms,” in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, - nil, nil. [Online]. Available: `https://doi.org/10.1109/ijcnn.2005.1556240`.

[37] B. Rueckauer, I.-A. Lungu, Y. Hu, and M. Pfeiffer, “Theory and tools for the conversion of analog to spiking convolutional neural networks,” *CoRR*, 2016. arXiv: `1612.04052 [stat.ML]`. [Online]. Available: `http://arxiv.org/abs/1612.04052v1`.

[38] D. Zambrano and S. M. Bohte, "Fast and efficient asynchronous neural computation with adapting spiking neural networks," *CoRR*, 2016. arXiv: `1609.02053v1 [cs.NE]`. [Online]. Available: `http://arxiv.org/abs/1609.02053v1`.

[39] J.-P. Pfister, T. Toyoizumi, D. Barber, and W. Gerstner, "Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning," *Neural Computation*, vol. 18, no. 6, pp. 1318–1348, 2006. [Online]. Available: `https://doi.org/10.1162/neco.2006.18.6.1318`.

[40] S. Bohte, J. Kok, and J. Poutré, "Spikeprop: Backpropagation for networks of spiking neurons.," vol. 48, Jan. 2000, pp. 419–424.

[41] D. Huh and T. J. Sejnowski, "Gradient descent for spiking neural networks," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., Curran Associates, Inc., 2018, pp. 1433–1443. [Online]. Available: `http://papers.nips.cc/paper/7417-gradient-descent-for-spiking-neural-networks.pdf`.

[42] R. Urbanczik and W. Senn, "A gradient learning rule for the tempotron," *Neural Computation*, vol. 21, no. 2, pp. 340–352, 2009. [Online]. Available: `https://doi.org/10.1162/neco.2008.09-07-605`.

[43] J. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in Neuroscience*, vol. 10, Aug. 2016.

[44] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks," *CoRR*, 2019. arXiv: `1901.09948 [cs.NE]`. [Online]. Available: `http://arxiv.org/abs/1901.09948v2`.

[45] F. Zenke and S. Ganguli, "Superspike: Supervised learning in multi-layer spiking neural networks," *CoRR*, 2017. arXiv: `1705.11146 [q-bio.NC]`. [Online]. Available: `http://arxiv.org/abs/1705.11146v2`.

[46] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic plasticity dynamics for deep continuous local learning (decolle)," *CoRR*, 2018. arXiv: 1811.10766 [cs.NE]. [Online]. Available: http://arxiv.org/abs/1811.10766v3.

[47] B. Scellier and Y. Bengio, "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation," *Frontiers in Computational Neuroscience*, vol. 11, p. 24, 2017, ISSN: 1662-5188. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncom.2017.00024.

[48] P. O'Connor, E. Gavves, and M. Welling, "Training a spiking neural network with equilibrium propagation," in *Proceedings of Machine Learning Research*, K. Chaudhuri and M. Sugiyama, Eds., ser. Proceedings of Machine Learning Research, vol. 89, PMLR, 16–18 Apr 2019, pp. 1516–1523. [Online]. Available: http://proceedings.mlr.press/v89/o-connor19a.html.

[49] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in Neuroscience*, vol. 9, no. nil, nil, 2015. [Online]. Available: https://doi.org/10.3389/fnins.2015.00437.

[50] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. D. Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha, "A low power, fully event-based gesture recognition system," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 7388–7397.

[51] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., Curran Associates, Inc., 2018, pp. 6571–6583. [Online]. Available: http://papers.nips.cc/paper/7892-neural-ordinary-differential-equations.pdf.

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

[53] K. Greff, A. Klein, M. Chovanec, F. Hutter, and J. Schmidhuber, “The Sacred Infrastructure for Computational Research,” in *Proceedings of the 16th Python in Science Conference*, K. Huff, D. Lippa, D. Niederhut, and M. Pacer, Eds., 2017, pp. 49–56.

[54] D. Heeger, “Poisson model of spike generation,” *Handout, University of Standford*, vol. 5, pp. 1–13, 2000.