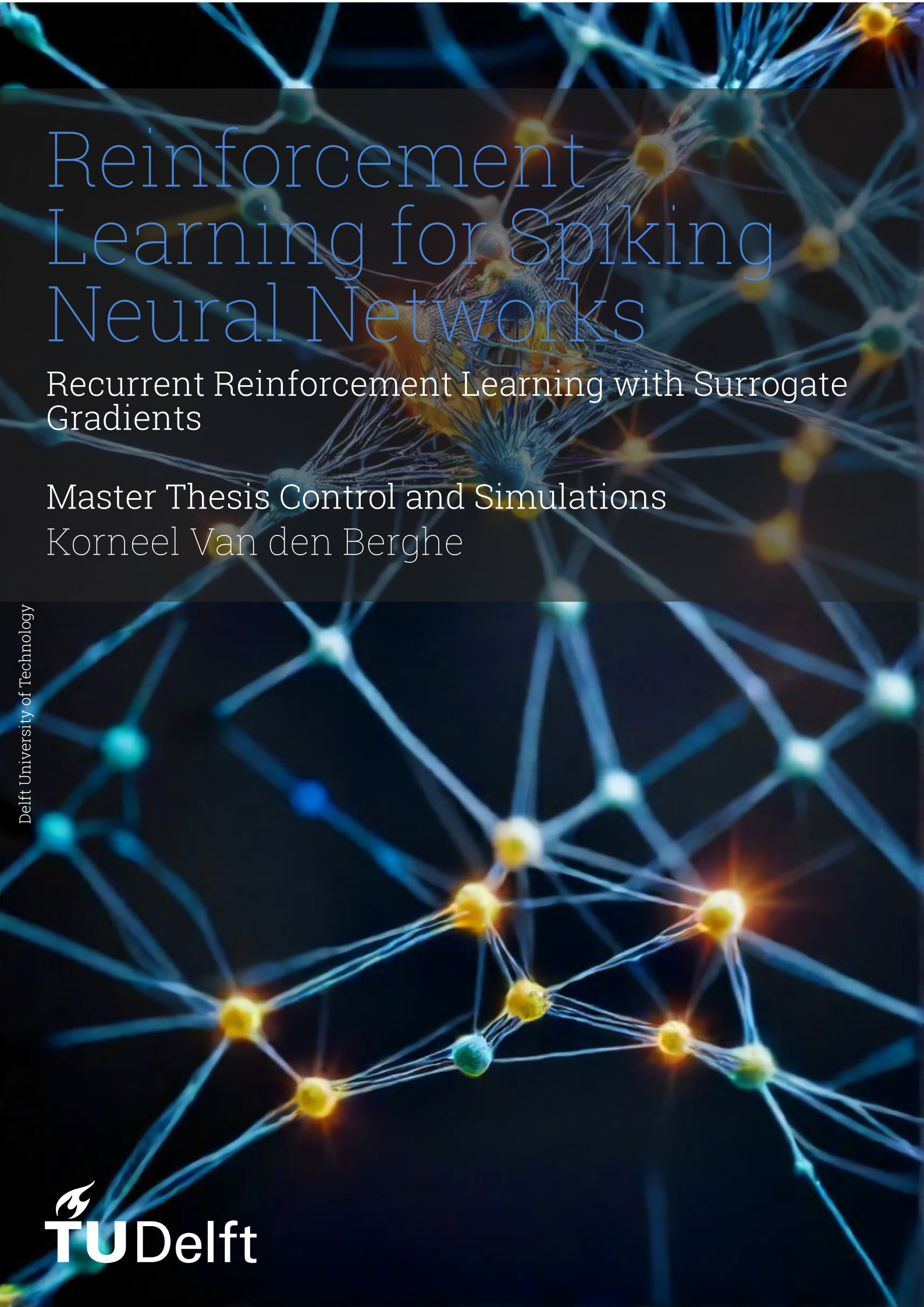


Reinforcement Learning for Spiking Neural Networks

Recurrent Reinforcement Learning with Surrogate Gradients

Master Thesis Control and Simulations

Korneel Van den Berghe

Delft University of Technology

TU Delft

Reinforcement Learning for Spiking Neural Networks

Recurrent Reinforcement Learning with Surrogate Gradients

by

Korneel Van den Berghe

Student Name	Student Number
Korneel Van den Berghe	5022878

Instructors:	G.C.H.E. de Croon, S. Stroobants
Project Duration:	March, 2024 - October, 2024
Faculty:	Faculty of Aerospace Engineering, Delft

Cover: Cover image generated by pollinations.ai with prompt: *Can you create a figure that displays the interconnection between reinforcement learning and spiking neural networks aiming to represent the temporal dynamics and temporal learning capabilities?*

TU Delft

Preface

I began my master's degree in September 2022. Although I had always told myself I would take a gap year between my bachelor's and master's to go surfing, I couldn't gather the courage to put my studies on hold to pursue this dream. This left me with a strange feeling—I no longer was fully enjoying my studies, yet I couldn't imagine doing anything else.

Things changed after receiving an email announcing the inauguration of Professor Guido de Croon. I had never attended an event such as this, nor thought I would ever attend one in the foreseen future, so I decided to go. Sitting in TU Delft's Aula, I found myself to experience an almost childlike excitement as Guido demonstrated the bio-inspired optical flow based drone landing. A new wave of curiosity and enthusiasm struck me.

Since then, I've been fortunate to explore my interests with incredible support along the way.

Firstly, I want to thank Professor Guido de Croon for his endless enthusiasm and guidance. Beyond supervising this thesis project, he supported me during my time as visiting researcher at Harvard University and helped me publish a research poster at ICNCE 2024.

I also owe gratitude to Stein Stroobants for his daily supervision, which allowed me to present this work. Thanks to his guidance, I've been able to navigate the world of neuromorphics and research. His kindness and helpfulness have made this thesis—and the projects leading up to it—a truly exciting experience.

I thank Professor Vijay Janapa Reddi for his supervision at the Edge Computing Lab, teaching me how to critically evaluate my work, refine my research focus and prepare for a future in academia. I extend my thanks to Erik van der Horst, who taught me flight testing performance and debugging real robotics.

To my peers at the Edge Computing Lab, for welcoming me into the lab and making my time in Boston a truly unforgettable experience, I am grateful. Similarly, to my MAVLab colleagues, with whom I spent countless late nights in the Cyberzoo, I thank the help, the laughter and the good pizza.

I thank my friends in Delft for standing by me over the past years, with whom I've made incredible memories—from coffee breaks to travelling the world together.

Lastly, I want to thank my family for supporting me in anything I do, have done and will do in the future.

I am lucky to have met all of you on my path.

Thank you.

Korneel Van den Berghe
Delft, December 2024

Summary

Reinforcement learning (RL) has emerged as a promising approach for achieving intelligence through experience, rather than relying on fitting models to datasets. This is particularly relevant in robotic applications, where learning from experience is crucial for developing true intelligence and enabling continuous training of robots in real-world deployments. However, current RL methods often assume the framework of Markov Decision Processes (MDPs), which may not align with real-world scenarios where rich temporal information is involved. To adapt to this, a time-history is typically appended to the state observations, but this introduces two key challenges. First, it increases computational demands due to the expanded input dimensions. Second, it requires prior knowledge of the specific time-history needed to effectively solve a task. Another challenge is the high computational complexity of artificial neural networks. Deploying these networks often necessitates expensive and heavy hardware, making them less energy efficient and limiting the scope of AI applications in robotics. Neuromorphic computing has emerged as a brain-inspired computing paradigm, which promises large energy efficiency gains. Spiking neural networks (SNNs) are a type of neural networks which use brain-inspired spiking neuron models rather than conventional activation functions. While this spiking behavior is largely responsible for their energy efficiency, they introduce a number of challenges while training. SNNs, inspired by the brain, use spiking neuron models to mimic biological neural communication. Unlike traditional artificial neural networks, which transmit information continuously, SNNs encode information in discrete spikes, akin to the electrical impulses found in biological systems. This design not only enhances the computational capabilities of individual neurons but also improves overall energy efficiency. By leveraging the timing and frequency of spikes, SNNs can process information more efficiently, making them particularly well-suited for resource-constrained applications like drone control, where energy consumption is a critical factor.

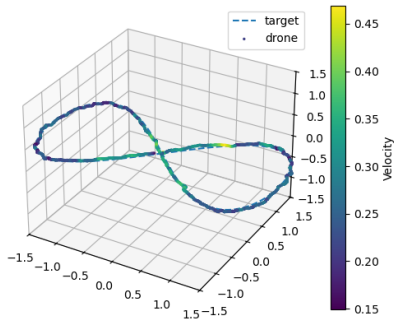
This report addresses temporal learning in RL and aims to establish an understanding of the effect of hyperparameters in spiking neural networks in an RL context. Next, a new reinforcement learning method is introduced. A method that integrates several RL concepts to enable the temporal training of deep spiking neural networks. An asymmetric actor-critic setup is used to train a spiking actor, but using a non-spiking critic for increased stability. By utilizing a privileged teacher actor, a neural network which can solve the task at hand with state information which is only available in simulation, one can roll in a spiking actor smoothly. The main advantage is that the sequences used to teach the spiking actor are now of usable length to learn temporal relations. Especially in situations where the length of an interaction is determined by the ability of the controller to actually solve the task, such as stabilization tasks. The method is specifically applied to an end-to-end drone control task. The task involves processing input data, including position, velocity, orientation, and angular velocity information, and generating the desired RPM settings for each motor.

In addition to proposing this method, the report compares two offline RL approaches: Twin Delayed Deep Deterministic Policy Gradient (TD3) with Behavioral Cloning (BC) and native BC, both of which use existing flight data. Finally, a network is trained using online TD3, which requires explicit state history at the expense of energy efficiency, reflecting the current approach of training SNN using RL. With the training framework in place, the potential of neuromorphic spiking neural networks (SNNs) is explored in detail. This analysis explores how SNNs can be integrated into the proposed RL framework, offering a promising solution to both computational complexity and energy efficiency challenges in drone control systems. It is found that the surrogate gradient settings, used to backpropagate through the network, plays an important role in RL. While this setting controls the scope of the weights that is updated, this comes at the cost of noisy training. Where in supervised training methods, injecting noise in training is usually undesirable, in RL, this can act as an exploration mechanism.

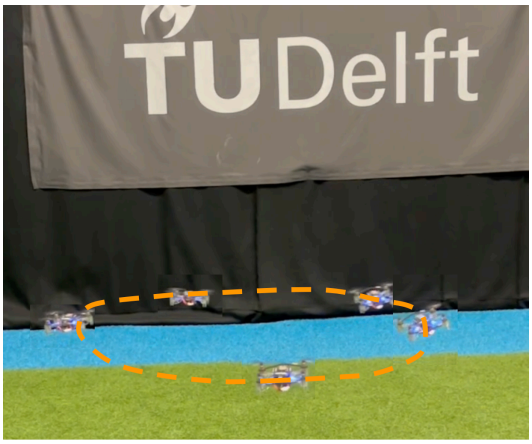
The results show that the proposed method, TD3+BC+JSRL, which combines elements from online TD3 and offline TD3+BC and bridges them using Jump-Start RL (JSRL), outperforms other approaches in terms of computational efficiency—measured using the NeuroBench benchmarking suite—and over-

all performance, as indicated by the total return achieved during flight. The spiking neural network is compared to existing solutions. The comparison reveals that while the spiking network achieves similar performance to the non-spiking alternative, it offers improved energy efficiency, albeit with reduced reliability.

Finally, the ability to bridge the reality gap is analyzed, as visualized on Figure 1. The trained network is deployed on the CrazyFlie drone. This 27 grams micro aerial vehicle serves as an excellent testbed for the proposed solution due to its restrained computational capabilities and its fast dynamics due to its size. It is found that the controller can successfully control the drone in hover and during maneuvers. The network, is then deployed on three different CrazyFlie setup, varying the propellers and motors, all of which can successfully fly.



(a) The controller can successfully navigate in a figure 8 in simulation.



(b) When deployed on the real CrazyFlie, the spiking network can successfully navigate circles.

Figure 1: Spiking neural networks trained with reinforcement learning can be deployed in the real world. On Figure 1a, the drone is flying in simulation, tracking an eight figure. On the right Figure 1b, the network is deployed on the Crazyflie and demonstrates flight in the real world.

Contents

List of Figures

Nomenclature

Abbreviations

Abbreviation	Definition
A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
ACER	Actor-Critic with Experience Replay
AI	Artificial Intelligence
ANN	Artificial Neural Network
BC	Behavioral Cloning
BPTT	Backpropgation Through Time
DDPG	Deep Deterministic Policy Gradient
DDQN	Double Q-Learning
DQN	Deep Q-Learning
DVS	Dynamic Vision Sensor
ES	Evolutionary Strategy
eProp	Eligibility Propagation
IMU	Inertial Measurement Unit
JSRL	Jump-Start Reinforcement Learning
LIF	Leaky-Integrate and Fire
MAV	Micro Aerial Vehicle
MDP	Markov Decision Process
PGQ	Policy Gradient with Q-learning
PPO	Proximal Policy Optimization
RDPG	Recurrent Deterministic Policy Gradient
RL	Reinforcement Learning
RPM	Revolutions Per Minute
R-STDP	Reward-modulated Spike-Timing-Dependent Plasticity
SAC	Soft Actor-Critic
SARSA	State-Action-Reward-State-Action
SNN	Spiking Neural Network
STDP	Spike-Timing-Dependent Plasticity
TD3	Twin Delayed DDPG
TRPO	Trust Region Policy Optimization
UAV	Unmanned Aerial Vehicle

Symbols

Symbol	Definition
a	Action
A	Advantage function
$\mathcal{A}$	Set of all actions
$b(s)$	Baseline in advantage calculation
D_{KL}	Kullback–Leibler divergence
$\hat{g}$	Sample-based estimate of the policy gradient
H	Time horizon of a task

Symbol	Definition
$H(\pi)$	Entropy of policy, π
I_{in}	Input Current
I_{syn}	Synaptic Current
m	number of samples
$Q(s, a)$	Q-value function, Q-value at state, s, for action, a
R	Reset Mechanism
$R(s', a, s)$	Immediate reward of the transition
$R(\tau)$	Return of trajectory τ
s	Spike in context of neuromorphics, state in context of RL
s'	Next state
$\mathcal{S}$	Set of all states
U	Membrane Potential
U_{thr}	Threshold Membrane Potential
$U(\theta)$	Loss as function of θ
$V(s)$	Value function, value at state, s
α	Synaptic Current Decay Factor
β	Membrane Potential Decay Factor
ϵ	Small threshold
γ	Discount factor for return calculation
π	Policy
ϕ	Critic parameters
θ	Actor or policy parameters
τ	Trajectory, a sequence of state, action, pars
ζ	Weighing factor

1 Introduction

The combination of AI and robotics has given rise to a new era of embodied intelligence, where machines are able to autonomously perceive, reason, and control their actions. Among the various approaches of AI, reinforcement learning (RL) has emerged as a very strong method for teaching agents optimal behaviors through interactions with an environment. It has demonstrated capabilities beyond human performance in solving tasks from strategic games, such as Go [65], up to control problems in robotics [32]. However, the computational requirement and energy inefficiency of traditional artificial neural networks (ANNs) pose vital bottlenecks for practical implementations, especially in resource-constrained scenarios such as aerial robotics.

Neuromorphic computing potentially provides a powerful means of overcoming these challenges by emulating the efficient and flexible mechanisms of information processing in the brain.

Spiking neural networks (SNNs), a critical component of neuromorphic architectures, mimic the unique spiking patterns of biological neurons. This mimicry enables energy-efficient processing and temporal dynamics that are well-suited for real-world robotic applications. However, integrating SNNs into RL frameworks poses unique challenges. These include the difficulties of training networks with non-differentiable spike functions and leveraging their temporal properties effectively. Existing approaches omit the temporal properties to leverage existing frameworks [10, 11, 74]. This thesis aims at these challenges by developing new methods of applying RL to SNNs, with a focus on the potentials of SNNs in providing end-to-end control in robotics tasks that rely clearly on temporal learning and are subject to severe energy efficiency constraints. The approach entails a new framework for the training of SNNs for drone control with both offline and online RL. The suggested framework utilizes an asymmetric actor-critic architecture that merges the stability property of non-spiking networks with the energy efficiency offered by spiking actors.

The contributions of this thesis are threefold. First, it makes a comprehensive analysis of the role of hyperparameters in SNNs in the context of RL and points out the trade-offs between stability, energy efficiency, and exploration. A novel hybrid reinforcement learning approach is introduced to reconcile the conventional and neuromorphic paradigms of learning. Finally, it confirms the proposed techniques with extensive experiments on the Crazyflie drone platform that also demonstrates the feasibility of employing SNN-based controllers in simulated and real environments. This work tries to advance the current state of the art in reinforcement learning for SNNs and open a pathway toward the general adoption of neuromorphic approaches in autonomous systems. Combining the convergence of biological inspiration and engineering principles helps to push forward sustainable, intelligent machines, which can operate effectively under more complex and dynamic environments.

First, an introduction to neuromorphics is described in chapter 2. This is followed by an overview of current RL approaches in chapter 3. This chapter serves as a primer to the theory and reasoning of the method proposed in chapter 4, which includes the scientific paper. Finally, three additional experiments are described in section 4.1.

2

Neuromorphic Computing

Modern computer architecture is largely based on the von Neumann architecture. Here, the processing and memory are separated into distinct blocks, requiring significant data transfer. This data transfer is largely responsible for the power consumption in modern computing [3]. Neuromorphic engineering originally aimed at building processing systems that emulate the bio-physics of neurons and synapses [41], bringing the compute and memory closer together. Today, the exact definition of neuromorphic computing remains a topic of debate within the neuromorphic community. However, in general, it encompasses either time-, event- or data-driven computation. For this work, we will focus on spiking neural networks (SNNs).

2.1. Spiking Neural Networks

Spiking neural networks (SNNs) are often referred to as the third generation of neural networks [37]. Unlike conventional artificial neural networks (ANNs), which communicate information through continuous values between nodes, SNNs convey information using discrete spikes or pulses over time. This distinction introduces unique concepts for modeling and learning in SNNs.

2.1.1. Terminology

Early research into these networks has largely come from the neuroscience community. Therefore, the terminology used in neuromorphic research often differs slightly from what is found in conventional machine learning research. For starters, within neuromorphics, the term synapse is often used to describe a connection between two neurons. A synapse connects a pre- and post-synaptic neuron, and transmits current from one neuron to the other.

In SNN, neurons are modelled as spiking units that emit discrete voltage spikes, or action potentials when their membrane potentials reach a threshold. This membrane integrates inputs from other neurons that are connected to the current neuron through synapses, as membrane potential. and transmits a current spike from one neuron to the other. The synapses have weights that determine the magnitude of influence a presynaptic spike has on increasing or decreasing the postsynaptic neuron's membrane potential.

2.1.2. Neuron Models

Where the learning of non-linear functions in ANN is enabled by continuous activation functions, SNN utilizes brain-inspired neuron models. These neuron models are usually described by a set of differential equations, where a membrane potential gets charged with incoming current and where the neuron outputs a spike in case the membrane potential exceeds a threshold. Many different neuron models have been demonstrated, ranging from computationally heavy accurate neuroscientific models, to less complex models that are commonly used in robotic and edge computing applications.

One popular and accurate neuron model is the Hodgkin-Huxley model, described in 1952 [31]. This model was originally developed to describe the mechanism for propagation of action potentials found

in a squid giant axon. It was rewarded with the 1963 Nobel Prize in Physiology or Medicine. This model closely resembled the observations from the squid giant axon, and was modelled with a simple electric circuit. However, this model is computationally complex to simulate and therefore is not commonly used in practical SNNs.

A widely used model is the Leaky-Integrate and Fire model (LIF). The LIF neuron is a first-order neuron model where the input current, I_{in} directly charges the membrane potential, U. This potential energy leaks over time at a rate β, the leakage parameter. When the membrane potential exceeds a threshold, U_{thr}, the neuron spikes and the membrane potential is reset. Several reset mechanisms are widely used, including soft reset, where the threshold is subtracted from the membrane potential, or hard reset, where the membrane potential is reset to zero after a spike. The charging and resetting of the membrane potential can be modeled using the following equation:

$$U[t+1] = \beta U[t] + I_{in}[t+1] - R \cdot U_{thr} \tag{2.1}$$

Where R is 1 whenever the membrane potential exceeds the threshold and 0 otherwise. The spiking behavior can be modeled as:

$$s = \begin{cases} 1, & \text{if } U[t+1] > thr \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

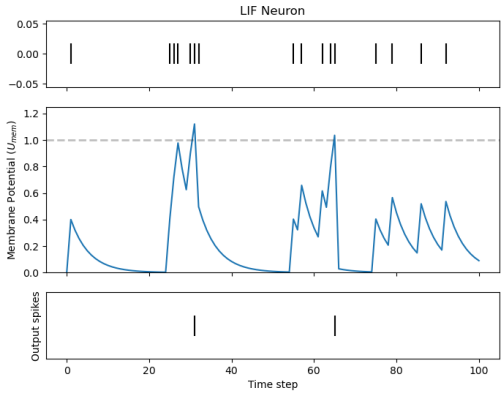
Next, second order neuron models such as synaptic models account for synaptic conductance. The synaptic current is charged by the input current, which subsequently charges the membrane potential, after which similar behavior as the LIF can be observed. Concretely, the following equations model these dynamics.

$$I_{syn}[t+1] = \alpha I_{syn}[t] + I_{in}[t+1] \tag{2.3}$$

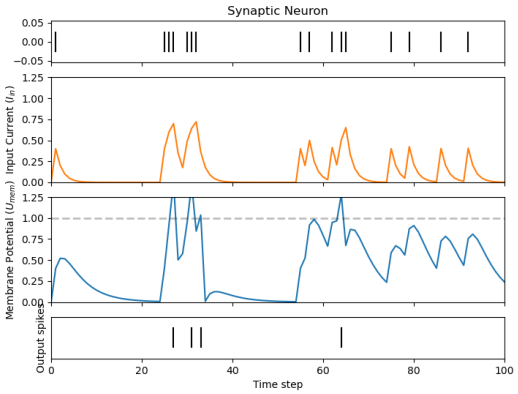
$$U[t+1] = \beta U[t] + I_{syn}[t+1] - RU_{thr} \tag{2.4}$$

The spiking behavior for this neuron is the same as the LIF neuron described above. Thanks to the decaying current, the second order model is able to learn longer time horizons.

To illustrate the effect of the decaying synaptic current, a simulation of both neurons is provided on Figure 2.1. An important difference between the first and second order neurons is the fact that the maximum membrane potential after an incoming spike occurs at the exact time of the incoming spike for first order models, while for second order models, this maximum is observed with a delay. Therefore, the effect of an incoming spike can accurately be timed to affect the output, where first order models will only indirectly carry information through time, due to the decay in membrane potential.



(a) Response of a LIF neuron to an incoming spike train.



(b) Response of a Synaptic neuron to an incoming spike train.

Figure 2.1: As can be seen on the figures, second order neurons allow for effects of inputs beyond its spike release, due to the leaking synaptic current.

2.1.3. Input Encoding and Output Decoding

Information processing within spiking neural networks is done through communication of spikes. While SNNs can process continuous values as incoming current, direct encoding, one often encodes the continuous values to spike trains and subsequently decodes the output spikes to continuous readings for further processing. This allows us to use neuromorphic accelerators that are optimized for spike information transfer. Encoding information to spikes can lead to significant energy efficiency gains as one would be able to exploit the full capabilities of popular neuromorphic accelerators, which often do not support continuous value information transfer. Various methods for spike encoding and decoding have been proposed [13, 70, 33, 2].

Rate Coding

Rate coding techniques are often employed in SNNs, where a continuous value is transformed into a sequence of spikes or a spike train. In this method, higher frequencies of spikes correspond to higher continuous values. One of the key advantages of this approach is its error tolerance. Even if a single spike is missing or misplaced in the spike train, it doesn't necessarily compromise the overall signature or pattern of the complete spike train. This resilience to minor errors can enhance the robustness and reliability of the network [17].

Temporal Coding

Temporal encoding is another method utilized in SNNs, where continuous values are transformed into specific spike timings. In this approach, the focus shifts from the quantity of spikes to the precise timing of a spike in a neuron. This shift in focus results in fewer spikes being required for data encoding, thereby making this method more energy-efficient. Additionally, temporal encoding tends to outpace rate coding in terms of speed. This is because it doesn't necessitate waiting for the completion of an entire spike train to identify the encoded value. Instead, the timing of individual spikes can provide immediate information, leading to faster data processing [17]. Furthermore, the usage of exact spike timing opens up another dimension of temporal processing.

Population Coding

Population coding is also widely used in SNNs [80]. In this approach, the collective activity of a group of neurons is used to represent an input signal. This method is advantageous due to its speed, as it allows for immediate encoding or decoding based on the spikes occurring at a specific time step. However, the range and precision of the encoding and decoding processes are reliant on the number of neurons available to represent continuous information. As such, this method may necessitate the use of large hidden layers to effectively capture and represent the complexity of the input data. This could potentially increase the computational complexity of the network, but it also allows for a more robust and detailed representation of the input signal.

2.2. Training Spiking Neural Networks

The discrete nature of the spikes that are outputted by the neurons raises an additional challenge. Therefore, the backpropagation algorithm [58], which has been central in the training of ANN, is not directly applicable to SNN. Alternative training methods have been explored, which can roughly be divided in three categories.

2.2.1. Local Learning Rules

First, there are local learning rules. Local learning rules change individual synapses, which are the connections between two neurons, based on a local rule. Often these rules are based on a principle originating from neuroscience research [5]. Hebbian learning [29] states that repeatedly and persistently co-active neurons should increase their connective strengths as a means of storing a memory trace, often explained by the phrase: *those who fire together, wire together*. Later, this rule evolved to what is currently known as spike-timing-dependent plasticity (STDP) [66]. STDP builds on top of Hebbian learning by introducing spike timing. When a pre-synaptic neuron spikes before the post-synaptic neuron spikes, the strength of the synapse is increased. However, when the post-synaptic

neuron spikes before the pre-synaptic neuron, the strength should be decreased. Other methods such as three-factor rules [19, 23, 60, 4] build further on these ideas. They introduce a third factor, such as the timing between the pre- and post-synaptic spike timing or an eligibility trace, which acts as an additional factor in the calculation of the weight change.

2.2.2. ANN to SNN Conversion

While backpropagation and local learning rules allow training SNNs directly, another common approach is to convert pre-trained artificial neural networks (ANNs) to spiking neural networks (SNNs) [53, 9, 16, 57]. This conversion process leverages techniques to train ANN, while taking advantage of SNN specific properties such as event-based communication between layers. A pre-trained ANN is converted by translating its weights and activations into a functional SNN. The continuous-valued activations are encoded into spike rates or spike timing for the SNN. Common encoding methods include rate coding where firing rates are proportional to activations, threshold coding where activations are converted to spike times by thresholding, and latency optimization which finds spike times to minimize latency.

2.2.3. Backpropagation in SNN

Third, there is backpropagation for SNN. Arguably the most popular backpropagation method for SNNs is the surrogate gradient technique [46]. Discrete spikes are used in the forward pass, but one approximates the spike function with a smooth, differentiable surrogate for the purposes of backpropagation. Typically, a smoothed version of the step function is used, such as the sigmoid or tanh functions. Where the spiking behavior of the neurons is governed by a step function that has a zero gradient for all values except the transition value, where it has an infinite gradient, Figure 2.2 shows that using a sigmoid as an approximation for this step function results in a bounded gradient.

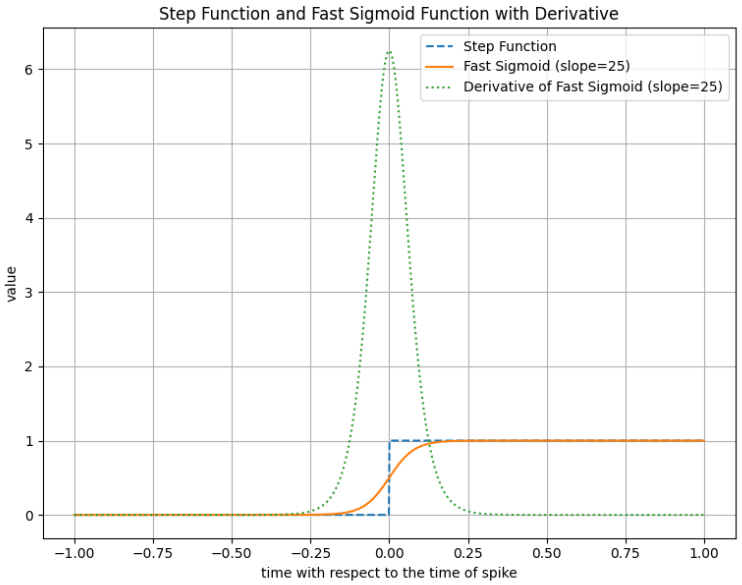


Figure 2.2: The step function, surrogate function and its gradient.

Eligibility propagation (eProp) [4], propagates the error signal from a loss function throughout the network. It accumulates effects of spiking activity over a temporal window. This allows to approximate the behavior of backpropagation through time (BPTT) and allows the network to maintain memory.

2.3. Neuromorphic Hardware

Next to the algorithmic developments, neuromorphic hardware has been proposed as well. This ranges from bio-inspired vision sensors [38] to accelerators for spiking neural networks.

2.3.1. Neuromorphic Accelerators

Neuromorphic accelerators implement neuron dynamics and synapses in dedicated analog, digital or mixed-signal circuits. Furthermore, accelerators exist for a wide range of scales to serve a wide range of applications. This ranges from cloud services, such as multi-chip platforms like Loihi [14] and SpiNNaker [40], to embedded sensing intelligent systems, such as Speck [67] and SNP [34]. While most neuromorphic accelerators focus on large energy improvement gains, by leveraging the sparse and, often binary, event-based communication in neuromorphic algorithms.

2.3.2. Event Cameras

Sensors that take inspiration from nature and work on an event-based principle allow us to incorporate them with neuromorphic algorithms and accelerators, while taking full advantage of the spiking properties. Proposed sensors range from sound sensors [62] to vision sensors [22]. For the purpose of this literature review, the focus is put on event-based vision sensors, which are most applicable to aerial robotics.

Event cameras are neuromorphic vision sensors, which has been a topic of research since the introduction of the integrated silicon retina by Misha Mahowald and Carver Mead [38]. They offer several advantages over conventional vision sensors. First, they operate continuously rather than frame based, as shown on Figure 2.3. Events are captured asynchronously, due to per-pixel brightness changes. As events are caused by per-pixel brightness changes, a sparse output is generated. Pixels that do not change, do not cause an event. The sensor tends to be very sensitive to brightness changes, offering a wide dynamic range, under a wide range of lighting conditions. Furthermore, the asynchronous nature of event-based vision allows a high temporal resolution compared to conventional sensors.
To truly take advantage of the asynchronous nature of event-based vision, novel algorithms and architectures have to be explored. However, their low-latency, low-power and high dynamic range capabilities have shown to be attractive for a wide range of applications [39, 75, 78, 77, 7]. Due to the high costs associated with event cameras, event based vision simulators have been developed, which simulate the inherently different workings of event cameras [55].

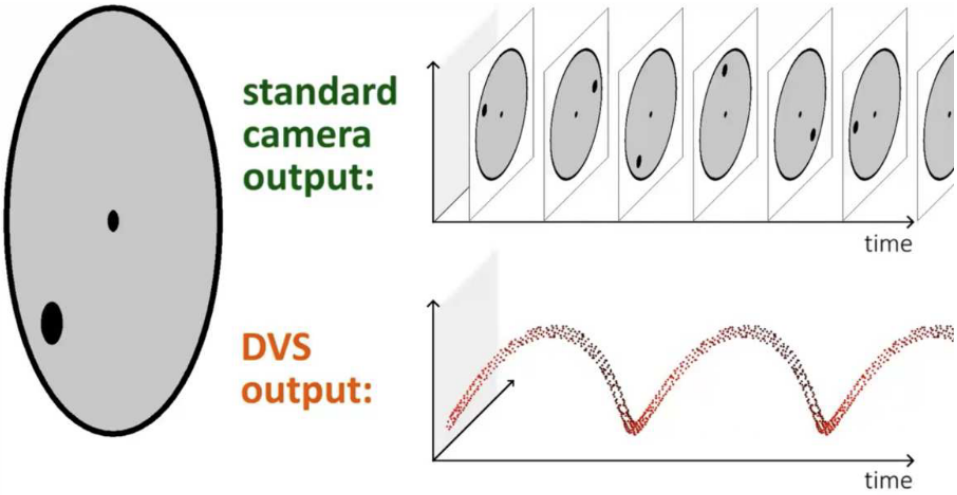


Figure 2.3: Standard camera output compared to event camera output. Where standard cameras capture frames at fixed time intervals, event cameras continuously stream changing pixel brightness. Therefore, the event camera does not suffer from motion blur, and only transmits motion data. The figure is taken from [45], and an animated version can be found on `www.youtube.com/watch?v=LauQ6LWTkxM`

2.4. Neuromorphics in Robotics

The energy efficient and dynamic properties of neuromorphics have gained the attention of the robotics community early on, trying to push the boundaries of embodied intelligence.

For extreme edge cases, such as micro aerial vehicles (MAVs) weight, energy and latency are major

concerns. Efficient compute leads to reduced battery requirements, which in turn leads to a decreased weight. Neuromorphics have been demonstrated for control of UAVs [69, 68, 75, 50]. It has been shown that basic controllers such as PID controllers can be made neuromorphic and can be used for onboard attitude control [69], this controller was not trained, but manually tuned. Next to controlling MAVs, it is possible to estimate the attitude of an MAV with an SNN in highly dynamic movements, using IMU data and an SNN with only 150 neurons [68], an impressive display of the temporal capabilities of SNN when one constraints and exploits the dynamics of the neuron models. This network was trained using a small dataset obtained with real MAVs. They were able to deploy the network on the Loihi [14] processor, paving the way towards energy-efficient, fully autonomous control of quadrotors. Establishing neuromorphic perception and control was demonstrated on a quadrotor using the Loihi [50]. The perception was based on optical flow and was trained in a self-supervised manner.

Other platforms have been demonstrated [6, 7, 8, 42]. Neuromorphic PID control has been demonstrated on blimps [8], where the SNN was trained rather than manually tuned. SNN has been used for lane-keeping tasks [7], where the controller was trained directly with R-STDP or indirectly, using conversion from an ANN. Other work explored autonomous racing, using techniques such as imitation learning or evolutionary learning for training [52, 81].

Next to SNN based planning, perception and control, neuromorphic sensors, such as event cameras, have found widespread adaptation in more conventional robotics as well [39, 22].

3

Reinforcement Learning

Finding the optimal set of parameters for a neural network for a specific task, requires numerous weight updates, which can be executed using algorithms such as backpropagation, local learning rules etc. In supervised learning settings, the error signal, which dictates the direction of a weight update, can easily be computed from datasets. In RL, however, there is no single correct output at each instance. The goal-driven nature leaves ample different solutions and approaches to achieve the final goal. Therefore, it is hard to compute a loss for each individual action, i.e. prediction. Numerous different methods to compute individual loss signals for each action have been proposed in RL. In this chapter, the basics of RL, an intuition behind the most popular algorithms, and finally previous work on RL for SNNs is discussed.

3.1. The Basics of Reinforcement Learning

Learning from experience is often desired when trying to train neural networks in situations where no clear examples exist. Popular applications for RL therefore include robotics, finance and natural language processing. The ability to learn from interaction with an environment, however, also opens up new challenges.

Most reinforcement learning algorithms use the logic described in Figure 3.1. An agent, a neural network in deep reinforcement learning, interacts with an environment by performing an action. The environment returns the next state, which subsequently is used as the input for the agent, and it returns a reward or penalty, which is used as an error signal to guide our agent during the learning process. The sequence of interactions, from start until the episode terminates, is called a rollout. During the learning process, one balances exploring the environment, by taking random actions or stochastic actions, and exploiting its skill, by following the agents' policy. The policy of an RL algorithm reflects the part of the agent that decides what next action to take, it is the part which will finally be deployed. Balancing exploitation and exploration allows the agent to not get stuck in local minima. All experiences are stored in a buffer, a data structure which holds the observations seen by the agent, the actions taken by the agent, the rewards received, and info about terminal conditions.

Reinforcement learning algorithms can be distinguished in several ways. Often, one distinguishes between on-policy and off-policy, one may also distinguish between value based methods and policy based methods, or offline and online reinforcement learning.

3.1.1. Online vs Offline Reinforcement Learning

When thinking about reinforcement learning, one usually imagines the scenario where the agent learns by directly interacting with a simulator or even with the real world. This setup, also displayed in Figure 3.1, refers to online reinforcement learning. However, alternatively, when demonstrations from an expert are available, one would like to exploit this data. Rather than using supervised learning, leveraging reward information enables learning the quality of state-action pairs in the dataset. This would be the offline reinforcement learning setup. It has been shown that offline reinforcement learning can

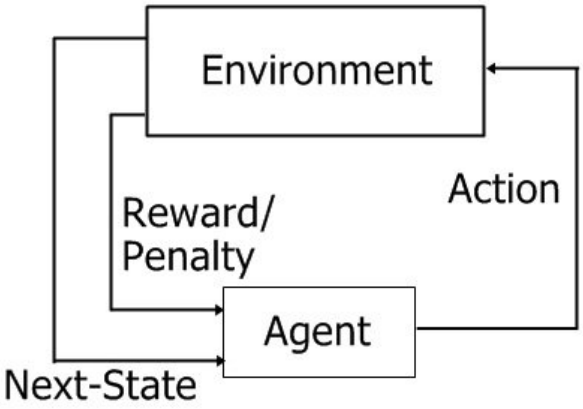


Figure 3.1: Diagram explaining the basic structure of most reinforcement learning algorithms.

successfully surpass the performance of the demonstrating dataset [21].

3.1.2. On-Policy and Off-Policy Learning

Within online reinforcement learning, there are, again, various ways to distinguish algorithms. Often, one performs several rollouts with the environment before performing an optimization step. One has the option to decide to keep rollouts gathered with a previous iteration of our policy, or to clear the buffer after every optimization step and fill the buffer with rollouts gathered only with our current policy. Referring to off-policy and on-policy RL respectively.

There are several advantages to on-policy algorithms. First, they tend to be more stable [71, 26]. Next, they are often easier to implement, requiring only rollouts of the current policy. Popular on-policy algorithms include State-Action-Reward-State-Action (SARSA) [71], Proximal Policy Optimization (PPO) [63], Actor-Critic Asynchronous Advantage Learning (A3C) [43], or its synchronous version Actor-Critic Advantage Learning (A2C) [43].

However, off-policy algorithms tend to be more sample efficient, as they can reuse samples gathered with previous policies [24]. Popular off-policy algorithms include Deep Q Learning (DQN), Double Q Learning (DDQN), Deep Deterministic Policy Gradient (DDPG) and Soft Actor-Critic (SAC).

Recently, algorithms that aim to combine the advantages of both on-policy and off-policy methods have been developed. Two basic approaches can be observed. Either, some ratio of on- and off-policy gradient steps to update the policy are carried out. Examples of this approach are the Actor-Critic with Experience Replay (ACER) [15] and the Policy Gradient with Q-learning (PGQ) [48] algorithm.

3.1.3. Value-based and Policy-based Methods

Within RL, most approaches try to optimize a value function, a policy or a combination of the two. Both have their advantages and disadvantages.

In value-based RL, a function is learned that approximates the value or expected return of observing a state or state-action pairs, reflecting value $V(s)$ and Q-value functions $Q(s, a)$ respectively. Intuitively, the network can learn a value off- and on-policy easily from experience, as an exact return is available for each step and therefore state-action pair in the replay buffer. These value estimates are used by the agent to make informed decisions about the best next action to maximize cumulative reward. In deep RL, Q-value functions, can be optimized to create a policy, as described in DQN [44] and DDQN [27]. They tend to converge faster and more reliably, in discrete action spaces. In continuous action spaces, it is challenging to predict the value of all possible actions.

Policy-based RL aims at training an agent to output an action given the observation, in a deterministic or stochastic manner. To train the policy, one commonly uses the policy gradient. In basic terms, the policy gradient increases the probability of actions in an interaction if the interaction was found to be successful, while doing the opposite if the interaction was found to be unsuccessful. As the policy outputs an action or a probability of a set of actions, they are more suited to continuous action

spaces. Furthermore, they are more suitable for stochastic tasks and allow for more stable training. In environments where the exact value of an action is hard to determine, such as a drone control task, policy based methods are preferred.

3.2. Introduction to Deep Reinforcement Learning Theory and Algorithms

In this section, the aim is to explain the intuition behind the development of various hallmarks in reinforcement learning algorithms. For a deep understanding of the basic concepts of reinforcement learning before the introduction of neural networks, the reader is referred to the book *Reinforcement Learning: An Introduction*[71]. For more details about each algorithm, the reader is referred to the original publication.

First, consider the general objective function of RL, which one aims to maximize, presented in Equation 3.1.

$$U(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau)] \tag{3.1}$$

Where

- θ: The parameters of the policy π_θ (such as the weights in a neural network) which are adjusted to maximize the objective.
- τ: A trajectory, or sequence of states, actions, and rewards generated by following the policy π_θ.
- π_θ: The policy function parameterized by θ, which maps states to actions.
- $R(\tau)$: The return of a trajectory τ, often calculated as the cumulative, discounted sum of rewards received along that trajectory.

In essence, this objective seeks to maximize the return obtained from a complete trajectory, which is generated using a policy π parameterized by θ. This objective can be exactly maximized using solution methods when the Markov Decision Process (MDP) is relatively simple. However, as the complexity of the problem increases, deep neural networks can be leveraged to solve larger, more complex MDPs.

3.2.1. Exact Solution Methods

In early reinforcement learning methods, where MDPs were relatively simple to solve, exact solution methods gained popularity. A major advantage of these exact solution methods is their proof of convergence.

Value Iteration

In value-based methods, the agent doesn't learn a policy (a direct mapping from states to actions) but instead focuses on learning the values of states or state-action pairs. These values provide insights into how beneficial it would be for the agent to be in a certain state or to take a specific action in a given state. The agent then chooses actions based on these value estimates, typically aiming to maximize the expected cumulative reward.

Value iteration is an exact solution method. In value iteration, one will learn the value of being in each possible state. Next, one can find the action which will result in the highest return using this function. The value $V(s)$ can be defined as Equation 3.2.

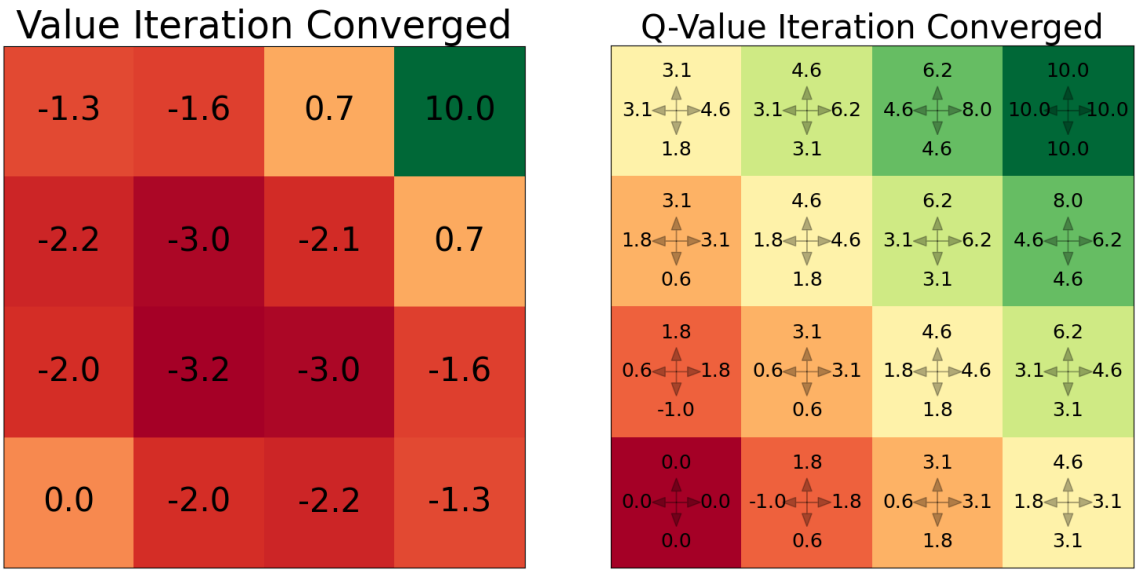
$$V(s) = \max_{a \in A} \sum_{s' \in S} P(s' \mid a, s)(R(s', a, s) + \gamma V(s')) \tag{3.2}$$

Where:

- $V(s)$ is the value function, representing the maximum expected cumulative reward obtainable starting from state s and following the optimal policy.
- $\max_{a \in A}$ indicates the maximization over all possible actions a in the action set A, ensuring the selection of the action that yields the highest expected reward.
- A is the set of all possible actions available to the agent in the environment.

- $\sum_{s' \in S}$ is the summation over all possible next states s' in the state space S, reflecting the stochastic nature of the environment.
- S is the set of all possible states in the environment.
- $P(s' \mid a, s)$ is the transition probability, representing the probability of transitioning to state s' from state s after taking action a.
- $R(s', a, s)$ is the immediate reward received when the agent transitions to state s' from state s after taking action a.
- γ is the discount factor, a parameter in the range $0 \leq \gamma < 1$ that determines how much future rewards are discounted compared to immediate rewards.
- $V(s')$ is the value of the next state s', representing the maximum expected cumulative reward obtainable from s' onward.

Looking at the example of a grid world, where an agent, initialized at the left bottom, is tasked to navigate to the right top of the world, one can visualize the values that are found using value iteration, displayed on Figure 3.2a.



(a) Value iteration predicts the value of being in each state. The optimal policy would equal to stepping towards the state with the maximum value.

(b) Q-value iteration will predict a value for each action in each given state. Therefore, the optimal policy would be to follow the largest Q-value prediction.

Figure 3.2: The values of the grid world example with a penalty of -1 for each step taken, a discount factor of 0.9 is visualized.

While value based methods can successfully solve MDPs, they require computing the value for each possible next state to decide on the best action to take. Alternatively, Q-functions receive a state and a proposed action, and will return the expected return of taking an action in a certain state. Mathematically, this alters Equation 3.2 to Equation 3.3.

$$Q(s, a) = \sum_{s' \in S} P(s' \mid a, s) \left(R(s', a, s) + \gamma \max_{a' \in A} Q(s', a') \right) \tag{3.3}$$

Applying Q-value iteration to the grid world example now outputs the Q-value function presented in Figure 3.2b.

Policy Iteration

Policy Iteration is an iterative method used to directly find the optimal policy. Unlike value iteration and Q-value iteration, which directly compute the value or Q-value function, policy iteration operates by alternating between two steps: policy evaluation and policy improvement.

Policy Evaluation

In the policy evaluation step, one computes the value function $V^{\pi}(s)$ for a given policy π. The value function $V^{\pi}(s)$ represents the expected return from state s when following the policy π. The Bellman equation for policy evaluation is given by:

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V^{\pi}(s') \right]$$

Where:

- $V^{\pi}(s)$: The value of state s under policy π.
- $\pi(a \mid s)$: The probability of taking action a in state s under policy π.
- $R(s,a)$: The immediate reward received by taking action a in state s.
- γ: The discount factor, $0 \leq \gamma < 1$.
- $P(s' \mid s,a)$: The transition probability of moving to state s' after taking action a in state s.
- $\mathcal{S}$: The set of all states.
- $\mathcal{A}$: The set of all possible actions.

The policy evaluation step is repeated until the value function converges, i.e., $\|V^{\pi}(s) - V^{\pi'}(s)\| < \epsilon$, where ϵ is a small threshold.

Policy Improvement

In the policy improvement step, one updates the policy by choosing the action that maximizes the expected return, given the current value function $V^{\pi}(s)$. The policy improvement step is performed as follows:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V^{\pi}(s') \right]$$

Where:

- $\pi'(s)$: The new policy at state s after the improvement step.

If the policy improves, the algorithm proceeds to the policy evaluation step with the new policy. Otherwise, the algorithm terminates as the optimal policy has been found.

The algorithm guarantees convergence to the optimal policy and value function.

3.2.2. Approximate Solution Methods: Value-based

While exact solution methods benefit from proven convergence and relatively simple implementation, they are limited to simple MDPs. Approximate solution methods in combination with deep neural networks have shown the ability to achieve impressive results on a wide range of tasks. Within the approximate solution methods, one can again distinguish in value- or policy-based methods.

Value-based reinforcement learning is commonly used in applications where discrete action spaces allow the agent to evaluate potential actions and select the one with the highest value. Practical applications include game-playing (such as AlphaGo and DQN for Atari games), robotics, and recommendation systems. Despite its simplicity and effectiveness, value-based RL methods are generally better suited for environments with manageable state-action spaces, as they can become computationally challenging in complex, high-dimensional spaces. For these reasons, ongoing research explores combining value-based techniques with deep learning, allowing them to scale to larger, more complex problems through methods like deep Q-networks (DQNs).

Deep Q Networks

One of the first displays of the true power of neural networks in RL came with the introduction of Deep Q Networks (DQNs), in 2013 [44]. In this paper, the authors presented the world with an algorithm which could outperform human players in a range of Atari games. The concept was rather simple. the agent approximates the *Q-value* of each state-action pair, by using deep neural networks. By iteratively updating Q-values using experiences gathered through interaction with the simulated environment, the agent improves its policy over time.

After sampling from the experience, stored in the replay buffer, one computes the target values, y using Equation 3.4:

$$y = \begin{cases} r & \text{if } s' \text{ is terminal} \\ r + \gamma \max_{a'} Q'(s', a'; \theta^-) & \text{otherwise} \end{cases} \tag{3.4}$$

Where $Q'(s', a'; \theta^-)$ is a target Q-function which is parametrized with θ^-), usually a delayed set of parameters from the most up-to-date Q-function. This serves as a stabilization mechanism for the learning process. While one could use the true return of a state-action pair, as stored in the dataset, this raises two challenges. Firstly, the true return is highly dependent on the exact sequence of state-action pairs, leading to high variance in the estimate. Secondly, to reuse experience which was gathered with an old policy, one needs to bootstrap with the most up-to-date Q-function. This allows to estimate the expected return, if one had followed the most up-to-date policy.

The loss, U, can be computed using Equation 3.5.

$$U(\theta) = \mathbb{E}\left[(y - Q(s, a; \theta))^2\right] \tag{3.5}$$

Double Q-Learning

While Deep Q-Networks (DQNs) marked a significant breakthrough in reinforcement learning by successfully approximating Q-values using deep neural networks, they suffered from an important issue: the tendency to overestimate Q-values. This overestimation arises because the same network is used both to select and evaluate actions, leading to overly optimistic value estimates and potentially suboptimal policies.

To address this issue, Double Deep Q-Networks (DDQN)[27] were introduced. The key innovation in DDQN is the decoupling of action selection and action evaluation, inspired by the Double Q-learning framework. Instead of directly using the Q-network to estimate the value of the best action in the target calculation, DDQN uses the primary Q-network to select the action and the target Q-network to evaluate its value. This modification reduces overestimation bias, leading to more stable learning and better performance.

The target value in DDQN is computed as:

$$y = r + \gamma Q'(s', \arg\max_a Q(s', a; \theta); \theta^-) \tag{3.6}$$

Where:

- $Q(s', a; \theta)$: The primary Q-network is used to select the action with the highest estimated value.
- $Q'(s', \cdot; \theta^-)$: The target Q-network evaluates the value of the selected action.

By introducing this separation, DDQN provides a more accurate estimate of the Q-value and enhances the robustness of the training process, especially in environments with noisy or stochastic rewards.

3.2.3. Approximate Solution Methods: Policy-based

In policy-based methods, one directly learns a policy $\pi(s)$. This is generally an easier policy to learn. Take for example a drone control task. The value of spinning up a propeller is rather ambiguous and is hard to learn, next to that, in robotics settings, high dimensional and continuous action spaces make value based methods challenging to use.

Vanilla Policy Gradient

Building on top of Equation 3.1, one can derive the policy gradient. First, replace the expected value by a sample-based estimate.

$$U(\theta) = \sum_{\tau} P(\tau;\theta)R(\tau) \tag{3.7}$$

Where

- $R(\tau)$ is the total return of a trajectory, τ, computed as $\sum_{t=0}^{H} \gamma^t r(s_t, a_t)$, H is the time horizon
- $P(\tau;\theta)$ is the probability of trajectory, τ under a policy parametrized by θ

To compute the gradient from Equation 3.7 from samples gathered in an environment, the equation should include a weighting term with the probability of a certain trajectory occurring. By realizing that the likelihood ratio introduces this weighting term, the following is derived:

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_{\tau} P(\tau;\theta)R(\tau) \\
&= \sum_{\tau} \nabla_\theta P(\tau;\theta)R(\tau) \\
&= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta)R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \nabla_\theta log(P(\tau;\theta))R(\tau)
\end{aligned} \tag{3.8}$$

Thanks to the weighting factor $P(\tau;\theta)$ introduced for each of the gradient computations, the final equation in Equation 3.8 can be approximated with samples following Equation 3.9.

$$\nabla_\theta U(\theta) \simeq \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta log(P(\tau^{(i)};\theta))R(\tau^{(i)}) \tag{3.9}$$

Note that $P(\tau^{(i)};\theta)$, the probability of following trajectory $\tau^{(i)}$ under a policy parametrized by θ can be rewritten as the product of state transition probability multiplied by the probability that a policy performs action, a_t given state, s_t, as described in Equation 3.10:

$$P(\tau) = \prod_{t=0}^{H} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t) \tag{3.10}$$

Where the state transition probability, $P(s_{t+1}|s_t, a_t)$, is independent of policy parameters θ. Therefore, Equation 3.9 can be temporally deconstructed to facilitate a gradient calculation for each step in a rollout. By taking the logarithm of the probability of a trajectory, one can take the gradient of the sum of Equation 3.10, and separate the state transition and the policy output in Equation 3.9. As the state transition is independent of the policy parameters, the gradient will be zero. While seemingly unimportant, this allows one to learn a policy with no knowledge of the dynamics of the environment, target platform or other state transition variables.

$$\hat{g} = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H-1} \nabla_\theta log(\pi_\theta(a_t^{(i)}|s_t^{(i)})) \sum_{k=t}^{H-1} \gamma^{k-t} r(s_t^{(i)}, a_t^{(i)}) \tag{3.11}$$

The expected value of the estimator $\hat{g}$, in Equation 3.9, matches the original gradient described in Equation 3.8, ensuring that it is unbiased. However, this estimator tends to have high variance. The key reason lies in its reliance on the exact returns obtained from individual trajectories. These returns depend heavily on the specific sequence of state-action pairs encountered during sampling. Since trajectory outcomes can vary significantly due to stochasticity in both the policy and the environment,

the resulting gradient estimates fluctuate widely. This high variance makes optimization inefficient, prompting the development of methods to reduce noise and stabilize the gradient estimates.

Advantage Estimation

In practice, one often subtracts a baseline, $b(s)$ from the true return, in an attempt to normalize the expected returns. Early approaches utilized constant baselines, time-dependent baselines, and even baselines derived from optimal control, all of which proved effective. The advantage can be calculated as:

$$\begin{aligned}\hat{A}_t^{\pi_\theta} &= \left[\sum_{k=t}^{H-1} \gamma^{k-t} r(s_t^{(i)}, a_t^{(i)})\right] - b(s_t) \\ &= Q^{\pi_\theta}(s_t, a_t) - b(s_t)\end{aligned} \tag{3.12}$$

Where the Q-value here is just the true return retrieved in the interaction, not a neural network. Which results in the final gradient calculation in Equation 3.13.

$$\nabla_\theta U(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t^{\pi_\theta}\right] \tag{3.13}$$

Advantage Actor Critic

The introduction of baselines significantly reduced the variance of policy gradient estimators without introducing bias. A choice for such a baseline is a state dependent value function $V(s)$. Next to using a neural network for the value estimate as baseline, one further reduces variance by bootstrapping the Q-value estimation with this critic, leading to the following advantage calculation.

$$\hat{A}_t^{\pi_\theta} = [r_{t+1} + \gamma V(s_{t+1})] - V(s_t) \tag{3.14}$$

In Advantage Actor Critic (A2C) or Asynchronous Advantage Actor Critic (A3C) [43], one uses such value function as a baseline. One initializes an actor network, π_{θ_0} and a critic network $V_{\phi_0}^{\pi}$, which predicts the value of being in a state and then following policy π. Both the actor and critic are then trained, using the collected rollouts, in an on-policy manner.

Trust Region Policy Optimization

One of the key challenges in RL is ensuring stable policy updates during training. It was soon realized that the magnitude of a gradient step can significantly influence an agent's performance. Unlike supervised learning, where an overly large step can often be corrected in subsequent updates, in RL, a large gradient step in the wrong direction can result in a suboptimal policy. This degraded policy can collect poor-quality data in subsequent rollouts, potentially leading to a complete loss of progress.

Additionally, when using advantage-based methods, the advantage calculated for a state-action pair under the current (or "old") policy may become irrelevant if the newly computed policy differs too greatly from the old one. This instability arises because the advantage values are tightly linked to the policy used to collect the rollouts.

To address these issues, Trust Region Policy Optimization (TRPO) [64] introduces constraints to stabilize policy updates. Instead of taking a single large gradient step, TRPO computes the gradient direction and applies multiple controlled updates to the policy. Using the importance sampling ratio, Equation 3.15, one can gain intuition that a policy update can be done by off-policy data, as long as this ratio is close to 1. Therefore, we can improve sample efficiency and reliably take multiple gradient steps without having to gather new rollouts.

$$\mathbb{E}_{x\sim P}[X] = \mathbb{E}_{x\sim Q}\left[\frac{P(x)}{Q(x)}X\right] \tag{3.15}$$

Using the importance sampling ratio, one can derive the policy loss function, taking into account the distribution shift of the policy when updating the parameters multiple times over the same data. Which

brings the following loss function:

$$U(\theta) = \mathbb{E}\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t^{\pi_{\theta_{old}}}\right] \tag{3.16}$$

Note that TRPO is an on-policy method, and $\pi_{\theta_{old}}$ refers to the policy which collects the experience, and is updated until the divergence becomes too large, and a new environment interaction is needed. To ensure that updates remain stable, TRPO constrains the Kullback-Leibler (KL) divergence between the old policy π_{old} and the updated policy π_{new}.

$$D_{KL}(\pi_{new}, \pi_{old}) \leq \epsilon \tag{3.17}$$

where ϵ is a small threshold that defines the trust region. By enforcing this constraint, TRPO prevents the new policy from diverging too much from the old one, ensuring that the advantage estimates remain valid, and the updates are reliable.

Proximal Policy Optimization

TRPO was a significant improvement compared to other advantage based methods, with increased stability. This came at the cost of computational complexity. With the introduction of Proximal Policy Optimization (PPO) [63], the idea was to bring the constraint into the loss function as a soft constraint, leading to

$$U(\theta) = \mathbb{E}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t^{\pi_{\theta_{old}}}\right] - \zeta\mathbb{E}\left[D_{KL}(\pi_{\theta_{old}}(\dots|s_t), \pi_\theta(\dots|s_t)) - \epsilon\right] \tag{3.18}$$

Where ζ is a weighing factor which can make this problem mathematically equivalent to TRPO, where the constraint is enforced in a hard manner. Due to this soft constraint, one is now able to leverage many deep learning tools and framework without significant changes. To reduce complexity, PPO V2 was introduced. Rather than using the KL divergence, a ratio $r_t(\theta) = \frac{\pi_\theta}{\pi_{\theta_{old}}}$ is introduced as a measure of the validity of using the advantage estimate. When this ratio diverges from 1, a new rollout should be collected. The loss function therefore reduces to Equation 3.19

$$U^{clip}(\theta) = \mathbb{E}\left[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right] \tag{3.19}$$

Deep Deterministic Policy Gradient

While the abovementioned policy-based algorithms have proven highly effective, they all assume the policy is stochastic and are designed for on-policy learning. This and the following algorithms will no longer be advantage based, but will learn directly through the critic. Deep Deterministic Policy Gradient (DDPG) [35] approaches the problem from another angle. Using a critic which now predicts a Q-value function, $Q(a, s)$, rather than the regular value function, $V(s)$, one can learn the policy solely through this Q-value function. Additionally, this allows for both stochastic and deterministic policies, in contrast to advantage based methods.

By using the critic directly in the loss function, the loss function can be formulated as Equation 3.20.

$$U(\theta) = \mathbb{E}\left[Q(s, \pi_\theta(s))\right] \tag{3.20}$$

This method allows for both deterministic and stochastic policies, but suffered from instability issues. A common failure case in DDPG is that the Q-value is being overestimated, encouraging the policy to take suboptimal actions, similar to DQN. Inspired by DDQN, Twin Delayed DDPG (TD3) aims to solve this issue.

Twin Delayed DDPG

In DDQN, the issue of value overestimation was solved simply by introducing a second critic, and trusting the minimum value of both critics that is being predicted. In Twin Delayed DDPG (TD3) [20], the authors propose a similar approach building on top of DDPG. Next to this value clipping, they also propose updating the policy at a lower frequency than the critics, leaving the critics to converge to more realistic Q-values before training the policy. Next, noise to the policy actions is added to prevent the critics to overfit on the policy actions taken. The loss function remains unchanged from Equation 3.20.

Soft Actor-Critic

Soft Actor-Critic [25] introduces an entropy term into Equation 3.20, rather than adding a second critic. Therefore taking a different approach than TD3. The entropy motivates exploration and prevents overfitting to the Q-values predicted by the critic. The loss function is modified to Equation 3.21.

$$U(\theta) = \mathbb{E}\left[Q(s, \pi_\theta(s)) + \zeta H(\pi(\dots|s))\right] \tag{3.21}$$

Note that the last three methods do not use advantage calculations or do not assume stochastic policies. Given the critic models the expected return for taking a certain action in a state and then using the current actor as policy, this thus also works off-policy, where advantage based methods inherently assume the policy does not tend too far away from the gathering policy.

3.3. Reinforcement Learning for Partially Observable Environments

A pitfall of common reinforcement learning methods is that they are trained on single transitions (consisting of an observation, action, reward and next observation). This raised two distinct challenges. First, one can ask how to train an algorithm to not only use spatial information, but use spatio-temporal or temporal information as well. Second, real life robotics often lack one or more states, which are available in simulations that are required to solve a task, called a partially observable environment. These missing states can often be derived from temporal information, think of velocity estimation from position information. Methods incorporating the temporal dimension have been proposed and demonstrated.

Only a few months after the groundbreaking release of the DQN paper [44], the algorithm was applied for a recurrent network [28]. Where the original DQN algorithm demonstrated human-level performance on several Atari 2600 tasks, it needed a buffer of 4 frames in order to make a prediction. The recurrent version of the algorithm showed to be effective with only a single input frame per prediction. Training these recurrent nets requires training on sequences of data rather than single transitions. However, DQN updates the networks based on the assumptions that all samples are independent and identically distributed, which is no longer the case when considering sequences. However, they found that even when violating these assumptions, the algorithm still delivered improved performance over the conventional DQN algorithm for several tasks.

Later, recurrent policies were applied to other off-policy algorithms as well. Using Recurrent Deterministic Policy Gradient (RDPG) [30], an agent can be trained that can not only integrate noisy sensory measurements, but can also utilize temporal information on longer timescales. These off-policy algorithms have further been explored using actor-critic networks such as SAC [76]. They found that using an off-policy actor-critic algorithm with a unique actor and critic, rather than sharing one or more layers between the actor and critic, deliver the best results. Out of several off-policy algorithms, including DDPG and SAC, it has been shown that the SAC algorithm delivers the best results for recurrent policies [47].

3.4. Reinforcement Learning for Spiking Neural Networks

As discussed in chapter 2, spiking neural networks have unique properties which pose unique challenges, but have attractive properties. In a reinforcement learning setting, indirect training techniques with ANN-SNN conversion [72, 51] have been popular. However, direct training using RL has been explored as well. This section gives an overview of current efforts toward direct reinforcement learning for spiking neural networks is discussed. The approaches can roughly be categorized in biologically inspired reinforcement learning, where local learning rules are guided by a reward signal, and classic reinforcement learning, where conventional RL algorithms are used with a spiking agent.

3.4.1. Conventional Reinforcement Learning

Leveraging proven reinforcement learning algorithms allows training SNN by backpropagating an error signal throughout the network. Several different algorithms, including DQN and A2C, have been modified in order to support SNN and train energy efficient algorithms. While current implementations successfully train energy efficient agents, they do not exploit the temporal capabilities of the biologically inspired neuron models. They use rate encoding techniques or present the input for several timesteps, making the networks converge to an action. This causes the influence of previous observations to fade away, together with its temporal information.

Off-policy algorithms such as the DQN algorithm, which is a value-based algorithm, have been used to train energy efficient agents acting on a wide range of tasks, from basic control problems such as the cartpole [1], to the Atari 2600 games [36, 10]. In DQN, an experience replay buffer stores state transitions (state, action, next state, reward). The value function learns to predict the discounted reward of taking an action given a certain state. As the prediction is based solely on the current state, no temporal dynamics can be learned [1]. Alternatively, sequences can be stored to train on temporal information, which has been shown for recurrent neural networks [28].

Next to value-based reinforcement learning, actor-critic networks have been explored as well. Spiking DDPG, an off-policy algorithm with a spiking actor, but a conventional critic, leverages the efficient learning of artificial neural networks to guide the training of the spiking neural network. Training the spiking actor with spatiotemporal backpropagation enables mapless navigation of a real robot [73]. While spatiotemporal backpropagation was used, due to the rate encoding, the temporal capabilities of the actor are limited. Furthermore, the algorithm fails to generalize to complex, high-dimensional environments. A similar method was used to control a 6 degrees of freedom robotic arm in a goal reach task [49]. Expanding on on-policy methods, a policy gradient learning rule using the first-to-spike and rate encoding enabled SNN to solve a windy grid task, reducing the number of spikes by an order of magnitude compared to ANN-SNN conversion [56]. Next, a population encoded actor demonstrated its applicability in a wide range of both on-policy and off-policy algorithms (PPO, SAC, DDPG and TD3) [74]. They demonstrated the actor to achieve a similar reward as their non-spiking counterparts in various continuous control tasks included in the OpenAI gym, tasks include the half cheetah, hopper, walker and ant. Other work has also demonstrated the effectiveness of population encoding combined with spatiotemporal coding of dynamic neurons, on the same tasks as before, again outperforming the non-spiking alternatives [79].

Across all proposed methods, the focus has been to train energy efficient agents. They demonstrated the ability of SNN to act in complex, high-dimensional environments with similar or superior performance to non-spiking counterparts. The temporal capabilities of SNN have so far not been exploited, as training has been performed on single transitions rather than on complete rollouts.

In previous work, the author tried to develop an on-policy RL method for training SNNs with RL. It was chosen to use A2C due to its relatively simple implementation and easy extension to training on sequences rather than single transitions. While the training was found to be successful for very basic tasks, such as balancing a cart pole, or landing a drone given altitude information, it struggled to generalize to more complex tasks. One issue was the instability raised by the spiking critic. Next to that, training was slow due to the limited parallelization available in the implementation. For more details, one can consult the supplementary materials section A.1 and section A.2.

3.4.2. Biologically Inspired Reinforcement Learning

As neuromorphic computing is usually biologically inspired, previous work has attempted to develop reinforcement learning methods that follow neuroscientific principles. These methods often use local learning rules to simulate processes such as dopamine in our brains.

Reward modulated STDP has been used to train a wide range of tasks. Early work showed STDP to train an agent that controls a worm to find food [18]. Grid search tasks have been trained using similar reward modulated methods [12]. Simulating the processes that teach the brain to learn have been demonstrated to enable the training of SNN in a biologically plausible way [59]. While many of these methods display temporal learning capabilities, they have only been demonstrated on relatively low-dimensional tasks. Finally, the SpikePropamine method [61] has shown impressive results on highly delayed reward tasks, where the temporal dimension is essential for successful completion. Furthermore, it was shown to successfully complete the half cheetah environment as well.

While traditional ANN training methods excel in spatial settings, these biologically inspired methods offer interesting possibilities in temporal learning.

4

Scientific Article

RECURRENT REINFORCEMENT LEARNING WITH SURROGATE GRADIENTS

K. Van den Berghe, S. Stroobants, V.J. Reddi, G.C.H.E. de Croon
Delft University of Technology, Harvard University

ABSTRACT

Enabling embodied intelligence in robotics presents several unique challenges. A first major concern is the need for energy efficiency, low latency, and strong temporal reasoning to facilitate effective real-world interaction. Neuromorphic computing has garnered attention as a potential solution to these problems. Secondly, when using deep neural networks, it is hard to shape a learning signal, due to the goal oriented nature of robotics. Reinforcement learning (RL) poses itself as a framework to leverage goal-directed reward functions to create this learning signal. A key challenge with recurrent and spiking neural networks trained via RL is achieving stable baseline performance, able to creating sequences long enough to stabilize hidden states. This stabilization is crucial for processing sequences that extend beyond the initial warm-up period of the temporal network. In this article, an online RL approach is proposed, enabling temporal training with minimal changes to existing online algorithms, introducing a secondary guiding policy whose sole objective is to prevent episode termination before the warm-up period is complete. This framework is demonstrated to outperform offline RL methods and significantly improve the wall clock time of online RL methods, adapted to sample sequences rather than single transitions. Next, the effect of surrogate gradients as a technique for translating the learning signal from the RL framework to weight updates is analyzed. It is found that the slope, parametrizing the surrogate gradient, plays a crucial role in online RL settings, and can be exploited as an exploration mechanism.

1 INTRODUCTION

Reinforcement learning (RL) has demonstrated the ability to teach non-linear function approximates such as neural networks to achieve superhuman performance on a wide range of tasks [21; 14; 39]. Many of these methods rely on the assumption that the process of state-action-next-state is a Markov Decision Process (MDP) [33]. Often, they can be formulated in such a way that they do follow this assumption, however in robotics, processes are often partially observable [37; 38], with rich information embedded in time. In practice, a history of states and actions is often added to the observation to allow the process to be modelled as an MDP [4; 17; 22; 6]. Stateful neural networks, such as recurrent (RNN) or spiking (SNN) neural networks, inherently are time-variant, leaving the desire to leverage their temporal capabilities to overcome the need for explicit frame-stacking.

SNNs have been praised as a brain inspired machine learning paradigm, with low computational costs, low latency and temporal computing capabilities, receiving great interest in the field of micro robotics [32; 3; 5; 24; 18; 41]. Naively training SNNs on temporal information with RL leads to noisy training and suboptimal performance. Firstly, conventional RL algorithms need to be adapted to accommodate training on sequences rather than on single transitions, which has been demonstrated in the past [13] for value-based RL. However, for tasks where subpar performance leads to early episode termination, achieving a baseline performance capable of generating sequences of considerable length for a significant learning signal to teach the network, is challenging. Secondly, the discontinuous spiking nature of SNNs prevents the application of regular gradient descent. The application of surrogate gradients in the backward pass [23], however, enables leveraging conventional deep learning frameworks for gradient calculation. The effect of approximating the true gradient with this surrogate in a continuous control RL setting, where the learning signal is a noisy estimate of the true learning signal, remains unexplored.

In this article, a new approach toward online RL on temporal information for stateful networks in continuous environments is proposed. This approach is compared to state-of-the-art offline and online RL approaches. We demonstrate the method using SNNs, but no assumption is made on the type of network in the RL pipeline. Furthermore, the effect of the surrogate gradients in different learning setups, namely offline and online RL, on a drone control task is evaluated. Next, the computational requirements of this temporally trained SNN is compared to single-transition trained SNN and conventional artificial neural networks (ANNs) using the NeuroBench benchmarking framework [40]. Finally, bridging the sim-to-real gap is demonstrated by deploying the trained agent to control the CrazyFlie drone.

2 Related Work

2.1 Spiking Neural Networks

SNNs are a class of neuromorphic deep learning algorithms, which leverage brain-inspired principles [19]. Most significantly for this article, where conventional ANNs make use of non-linear, time-invariant activation functions with continuous outputs, SNN use brain-inspired neuron models that are time-variant and display a spiking behavior. This spiking behavior is mathematically governed by a Heaviside step function, for which the derivative is a Dirac Delta. This function is 0 almost everywhere, but at the origin, where it reaches infinity. Therefore, it leads to an extremely sparse and unstable gradient calculation. This calls for alternative methods for training. Proposed approaches have used alternative signals such as spike-timing to compute gradients, or used local learning rules [7; 15; 27; 20; 2], inspired by neuroscientific findings. In this article, the surrogate gradient technique [23] is used, to enable leveraging existing RL methods. The Dirac Delta is replaced by the derivative of a smooth surrogate function, which is parametrized by a slope k. The introduction of the surrogate gradient introduces bias to the gradient estimate, and can lead to sign reversal of the gradient in deeper networks [11].

2.2 Reinforcement Learning for Stateful Netwoks

Deep RL traditionally uses single observation, action, reward, next observation transitions during training. However, in real-life scenarios, important information can often be embedded in the temporal dimension. This issue was recognized early on and resolved by the use of frame-stacking [21], where a history of observations is passed to the actor for each prediction. R2D2 [13] directly trains RNNs on sequences, omitting the need for frame-stacking. The network is given a warm-up period during which its actions are not evaluated, to allow the hidden states to converge to realistic inference conditions. Then, the network is evaluated on the remaining observations from the sequence. This value-based method is ideally suited for discrete action spaces where the episode length is not proportional to actor performance. Jump-Start RL (JSRL) [36], originally proposed for improved exploration, can be used when an existing controller is available to gather lengthy sequences. Furthermore, when pre-acquired data is available, offline RL methods [9] can outperform the demonstrating policy, by exploiting reward information.

In the context of SNNs, both conventional as bio-inspired RL methods have been used for training. Bio-inspired RL methods [8; 28] often struggle with generalizing to bigger networks of complex tasks, and do not utilize the deep learning frameworks readily available. When leveraging traditional RL methods, the time-variance of SNNs is commonly circumvented by allowing each input to be processed multiple times [4; 1; 18; 35; 34]. This leads to final task performance in the same range as their non-spiking counterparts, but at the cost of computational efficiency and disregarding temporal relations. In actor-critic setups, an asymmetric setup improves performance. The critic, responsible for guiding the actor during training, is an ANN [35]. Next to asymmetry in model types, asymmetric inputs between actor and critic, where the critic exploits all information in the simulator, which is not available to the actor when deployed, tends to improve training [25; 6].

3 METHODS

3.1 ONLINE SEQUENTIAL REINFORCEMENT LEARNING FOR CONTINUOUS ACTION SPACES

In online RL, the actor is tasked with learning a behavior based on interactions with a simulated or real world environment, guided by a reward function. A key consideration when training stateful neural networks, such as spiking or recurrent networks, is gathering sufficiently long sequences to enable efficient training, allowing a warm-up period. For tasks where the length of an interaction is governed by the actors' performance, it takes a significant effort to achieve such baseline performance in online RL settings. Next, where recurrent RL methods has been demonstrated on value based methods, policy based RL methods are more suitable for continuous action spaces.

Within policy gradient methods which use the actor-critic setup, two approaches exist. Either the critic is used directly to compute the expected return, such as in DDPG [16], TD3 [10] or SAC [12], or the critic is used as a baseline, enabling advantage computation of the performed actions, such as in A2C, A3C [22], TRPO [30] and PPO [29]. As highlighted in previous work [38], algorithms which perform policy gradient using a baseline or an advantage estimate call for a baseline which accounts for the temporal characteristics of the policy. To simplify the algorithm, it was decided to use the critic directly to compute the expected return, removing the temporal dependence of the baseline . Lastly, even though the actor is time-dependent, it can be proven using the importance sampling ratio, that this time-dependence does not inflict with any assumption made in the derivation of the policy gradient (subsection A.3), on which many modern RL algorithms build.

3.1.1 LEVERAGING JUMP-START RL

In Jump-Start RL [36], a pretrained guide policy is used to gather a wide range of start conditions, after which an exploration policy is used for the remaining environment steps. While originally proposed for enhanced exploration, it poses itself suitable for training stateful networks, thanks to its inherent warm-up period. As in reality, online RL is almost exclusively performed in simulation, the guide policy can receive any set of observations from the simulator, similar to the asymmetric actor-critic setups that have gained popularity. Furthermore, the sole purpose of the guiding policy in this setup is avoiding termination before a number of warm-up steps of the stateful exploration policy. This guiding policy therefore does not need to be an expert policy, and can even be privileged. The guiding policy used, is a regular feedforward ANN, which receives the same privileged inputs as the critic. The exploration policy is a SNN.

Consider the environment to have a limit of 500 timesteps before it is terminated. Either, the guide controller is used for the first $500 - N$ timesteps, after which the exploration policy interacts for the remaining N steps, gradually increasing the N steps gathered with the exploration policy, until the guide policy is only used for the warm-up period. Alternatively, the guiding policy is used exclusively during the warm-up period.

3.1.2 SAMPLING STRATEGY

In the setup used in this article, the full rollouts, i.e. interactions from both the guiding as the exploration policy, are stored sequentially in the replay buffer. For training, sequences of length 100 timesteps are sampled from this buffer. The stateful policy is allowed a warm-up period of 50 timesteps, and actions are evaluated for the remaining timesteps. For ease of implementation, the hidden states are not stored, but initialized at zero at the start of the warm-up period. Alternatively, the hidden states could be stored and initialized at each sampled sequence, leading to a marginal increase in training performance [13]. The critic, which is not stateful, performs an update on all 100 timesteps.

3.1.3 LEARNING THE ACTOR

The learning approach for this unique setup should leverage the demonstrations from the guiding policy, as well as the exploration from the exploration policy. The buffer is partially filled with interactions from the guide policy, which reflects an offline learning setup, as well as interactions from the exploration policy, for which a classic online RL approach is expected to behave well. It has been shown that appending a behavioral cloning (BC) loss term to an online RL algorithm

allows for surpassing dataset performance in offline settings [9]. Therefore, we make use of a loss function which implements a weighted BC term that gradually decays during training, leveraging the demonstration data efficiently, but at the same time allowing the use of the actor-critic setup to exploit the reward signal as learning signal. In cases where the guiding policy is an expert policy, given its ability to exploit all properties of the simulation in its observation, one can choose to increase the weight of this BC term. Decaying this term becomes more important as the buffer is filled with more and more data from the exploration policy through training. Note that the BC term also serves as a soft constraint to avoiding large changes in policy behavior over the past policies, where large changes can discard the usability of off-policy data.

The update rule can be defined as:

$$L_\pi = -\mathbb{E}_{\tau\sim\mathcal{D}}\left[\sum_{i=0}^{100} Q_{\phi_1}\left(\mathbf{s}_{\tau,i}, \pi_\theta(\mathbf{s}_{\tau,i}|(\mathbf{s}_{\tau,i-1},\ldots,\mathbf{s}_{\tau,i}))\right)\cdot \mathbf{1}_{\mathrm{i}>49}\right] + \lambda\mathbb{E}_{\tau\sim\mathcal{D}}\left[\sum_{i=0}^{100} \|\pi_\theta(\mathbf{s}_{\tau,i}|(\mathbf{s}_{\tau,i-1}...\mathbf{s}_{\tau,i})) - \mathbf{a}_{\tau,i}\|^2 \cdot \mathbf{1}_{\mathrm{i}>49}\right] \tag{1}$$

Where:

- Q_{ϕ_1} in the first term is the first critic network used in TD3, which can essentially be replaced by any online RL policy loss.
- τ represents a sequence sampled from the replay buffer, of length 100. $\mathbf{s}_{\tau,i}$ and $\mathbf{a}_{\tau,i}$ are the i^{th} observation and action in the sampled sequence, respectively.
- λ is a hyperparameter controlling the strength of the BC regularization, decaying over time. When the guiding policy is of high quality, one would prefer a larger λ.
- $\mathbf{1}_{\mathrm{i}>49}$ is zero as long as the warm-up period of 50 timesteps has not passed, else 1.

3.1.4 Learning the Critic

The critic used during training is not time variant and thus does not need a warm-up period. Therefore, single transitions are sampled, and a traditional update rule can be used:

$$L_{Q_{\phi_i}} = \mathbb{E}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}',\mathbf{a}')\sim\mathcal{D}}\left[\left(Q_{\phi_i}(\mathbf{s},\mathbf{a}) - y\right)^2\right] \tag{2}$$

where $i \in \{1,2\}$, for the two critics used in TD3, and the target value y is computed using the Bellman equation:

$$y = r + \gamma \min_{j=1,2} Q_{\phi'_j}(\mathbf{s}', (\mathbf{a}' + \epsilon)) \tag{3}$$

- Q_{ϕ_i} and $Q_{\phi'_j}$ are the critic networks (with ϕ'_j being a delayed target network).
- $\mathbf{s}$ is the current state, $\mathbf{a}$ is the current action, and r is the reward.
- $\mathbf{s}'$ is the next state, and $(\mathbf{a}'+\epsilon)$ is the action from the transition with added noise $\epsilon \sim \mathcal{N}(0,\sigma)$ is small random noise for smoothing the target policy (as part of TD3's delayed policy update).
- γ is the discount factor.

Note that in practice, the target value, Equation 3, in TD3 or SAC is usually computed by computing the next action from the next observation using the current policy, $\pi_{\theta'}(\mathbf{s}' + \epsilon)$. We sample it from the transition, avoiding recomputing the next actions, omitting an additional warm-up period for stateful actors. While this will bias our critic towards learning the expected return from the actor which gathered the transition, it is assumed that our current policy did not diverge from the gathering policy significantly, controlled by the BC term.

3.2 Spiking Actor-Critic

The spiking policy uses two hidden neuron layers, with Leaky-Integrate and Fire neurons (LIF). An input current, I_{in} to the neuron charges the membrane potential, U, which leaks away over time, the

rate of which is defined by β. When the membrane potential exceeds a threshold, the neuron outputs a spike and the membrane potential is reset. Which can be computed using:

$$U[t+1] = \beta U[t] + I_{in}[t+1] - R \cdot U_{thr} \tag{4}$$

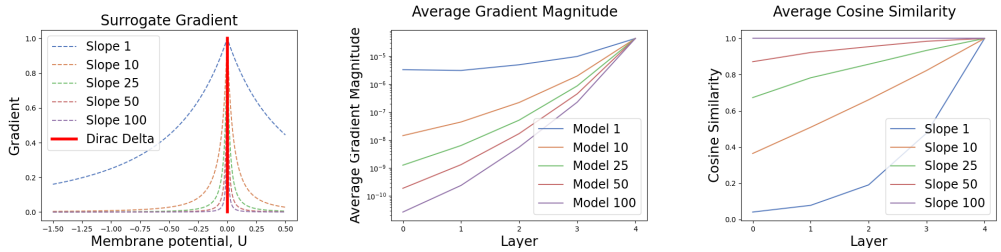
Where R is 1 whenever the membrane potential exceeds the threshold, U_{thr} and 0 otherwise. The spiking behavior can be modeled as:

$$s = \begin{cases} 1, & \text{if } U[t+1] > U_{thr} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

Note that this is a first order model, which means that a large input essentially immediately causes a spike to be generated. In situations where an incoming spike should affect a neuron at a later time step, second order models could be more effective.

The inputs to the network are the position, linear velocity, orientation (as rotation matrix) and angular velocities, and the network outputs motor commands. The inputs are encoded using a linear fully connected layer, and the output is decoded using population encoding. While the actor is spiking, the critic is parametrized using an ANN, which receives the aforementioned states and an action history of 32 timesteps. As this critic is not deployed on the drone, the deployed controller will still be a fully spiking network. Furthermore, this asymmetric setup has been demonstrated to successfully leverage the improved training stability of ANN [35; 34].

3.3 Surrogate Gradients



(a) The slope of the surrogate gradient dictates the range of inputs for which a gradient exists. The real gradient of the step function is the Dirac Delta.

(b) A more shallow slope carries the gradient through deeper through the network, therefore suffering less from the vanishing gradient problem.

(c) A shallower slope introduces bias and variance to the estimator. In deeper layers, the cosine similarity of the true and surrogate gradient goes to zero.

Figure 1: The effects of surrogate gradients on the gradients computed for deep networks.

As explained in subsection 2.1, surrogate gradients allow one to calculate a gradient for SNNs. In spiking networks, the input to the step function, which is replaced by a surrogate function in the backward pass, is the membrane potential, $U \in \{0, U_{thr}\}$, where U_{thr} is the threshold membrane potential. The step function is centered around the threshold, leading to inputs $x \in \{-U_{thr}, 0\}$. Using a surrogate gradient with a shallow slope allows a larger range of inputs to have non-zero gradients, as seen on Figure 1a, increasing the number of weights updated during each backward pass, as reflected in the gradient magnitude in Figure 1b. The steeper the slope, the bigger the effect of the vanishing gradient. The vanishing gradient is a well known issue in machine learning, which is emphasized in SNNs, as a true gradient only exists when the spike occurs.

While the slope of the surrogate gradient affects the number of updates done at each backward pass, it also introduces bias and variance[11]. Using the cosine similarity of two vectors, one can analyze the similarity of the true weight gradient $\nabla \mathbf{W}_i$ and the weight gradient computed with a surrogate gradient, $\tilde{\nabla} \mathbf{W_i}$.

$$\text{cosine similarity} = \frac{\nabla \mathbf{W}_i \cdot \tilde{\nabla} \mathbf{W_i}}{\|\nabla \mathbf{W}_i\| \|\tilde{\nabla} \mathbf{W_i}\|} \tag{6}$$

As the true gradient for deeper layers does not exist, the cosine similarity of the weight update computed with the surrogate and true gradient is analyzed by comparing a shallow slope to a steeper

slope. More specifically, the cosine similarity of a slope 100 is used as the *ground truth*. When the slope approaches the *true* gradient of 100, the cosine similarity remains close to, 1 even for deeper layers. For shallow slopes, it is seen on Figure 1c that the cosine similarity reduces to 0, indicating that the weight update is essentially a random vector. This will essentially break the training for deeper networks.

In RL, it is important to balance exploration and exploitation, meaning taking random actions, to increase the number of visited states, and using the current policy to compute the next action. This allows the policy to escape local minima and converge to an optimal policy. Shallow surrogate gradients, naturally introduce randomness in the weight update. This hints towards the possibility of leveraging this as an exploration mechanism in off-policy learning, where the randomness would be captured in the replay buffer. One could also schedule this surrogate gradient slope, to increase in steepness, as training progresses. While this showed to improve performance in initial experiments, it was found this improvement is highly sensitive towards the used schedule. Therefore, scheduling was discarded for the subsequent results.

3.4 Simulated Environment

To demonstrate the ability of the spiking network to bridge the reality gap, the CrazyFlie drone is used as the target platform. As a lightweight and computationally constrained device, it serves as an excellent testbed for the proposed training methods. Due to the drone's inherent instability, it presents itself as an ideal platform to analyze the proposed learning algorithm Inadequate controllers cause early crashes, resulting in unusable training data. Additionally, its limited processing power, high control frequency, and stringent energy demands place micro aerial vehicles in a category of robotics that could significantly benefit from neuromorphic computing solutions.

Training online RL algorithms using a real drone, however, is challenging due to the expensive data retrieval process. Learning therefore happens in a simulated environment, using a dynamics model which has demonstrated sim-to-real bridging capabilities [6]. The actor is penalized for errors in position, linear velocity, orientation, angular velocities and magnitude of actions. A more detailed description of the environment and reward structure can be found in subsection A.2.

4 Results

A key consideration when training stateful neural networks, such as spiking or recurrent networks, is gathering sufficiently long sequences to enable efficient training, allowing a warm-up period. When using online reinforcement learning algorithms, it takes a significant effort to achieve such baseline performance. Therefore, alternative methods which do not suffer from this issue, namely BC, TD3+BC are compared to the proposed method. When a learning signal from the RL framework is calculated, one can use the surrogate gradient technique to compute the required weight changes. Due to the notorious sensitivity towards hyperparameters in RL, one needs to establish an understanding of the effect of surrogate gradient slopes as a hyperparameter and whether the noise inducing effect of the shallow gradient can be balanced by the increased number of weight updates for each backward pass.

4.1 Surrogate Gradients as Exploration Mechanism

To investigate the effect of surrogate gradients, both BC, which is close to supervised learning, and TD3, an online, off-policy RL algorithm, are compared. In BC, distinct behaviors can be observed for the optimal surrogate gradient slope, for specific model sizes, displayed in Table 1.

For smaller models, improved model performance can be observed when using a shallow slope. As observed in Figure 1b, the scope of weights that are updated increases with shallower surrogates, as well as the magnitude of the update. In SNNs, a small change to the connecting weights does not necessarily lead to different outputs in the forward pass. The change in membrane potential might not affect whether the threshold governing the spiking behavior is exceeded. This also means that dead neurons, neurons that never spike, might remain dead, and saturated neurons, neurons that constantly spike, may remain saturated. When the model capacity tends to the limit of what is required for the problem to be solved, dead or saturated neurons essentially decreases the model

Best Test Performance after 500 Epochs				
	Slope 2	**Slope 50**	**Slope 100**	**Slope Scheduled**
16-16	40.1 ± 26.6	21.4 ± 1.7	21.2 ± 3.1	30.6 ± 2.0
32-32	99.3 ± 66.6	110.4 ± 49.8	30.4 ± 32.8	127.5 ± 63.0
64-64	124.6 ± 81.0	220.6 ± 89.8	172.2 ± 82.7	206.2 ± 83.8
128-128	276.9 ± 90.6	346.6 ± 95.0	275.8 ± 96.0	307.8 ± 91.9
Epochs to Surpass Mean Return of 100				
	Slope 2	**Slope 50**	**Slope 100**	**Slope Scheduled**
16-16	N/A	N/A	N/A	N/A
32-32	466 ± 189	N/A	N/A	466 ± 181
64-64	241 ± 147	53 ± 230	93 ± 119	68 ± 181
128-128	52 ± 52	25 ± 24	52 ± 24.5	29 ± 17

Table 1: Training performance across various model sizes and surrogate gradient slopes in a fully supervised behavioral cloning setup. Performance is measured by the return achieved when deploying the trained model in simulation. Training is capped at 500 epochs, with models using scheduled or slope 100 settings showing continued improvement, while shallower slopes converged early, displaying no further gains in the final epochs. Green cells represent the best performing algorithm, orange cells represent the second-best algorithm.

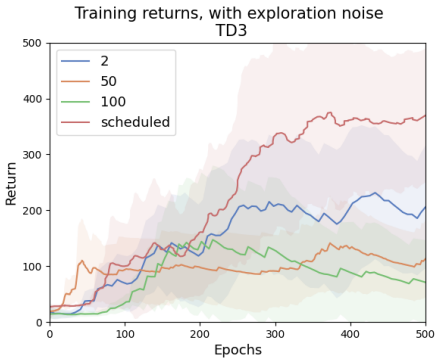
capacity further. Larger models have sufficient neurons such that the reduced number of updated weights is balanced by the more accurate gradient step. For larger models, it is found that there is an optimal to be found in slope setting, leading to both faster convergence and better final performance. It is however important to note that the training was stopped at 500 epochs. In the training curves it can be seen that the slope of the curves tend to 0 for the shallower surrogate gradients, while the surrogate gradients, that are scheduled or were parametrized with a slope of 100, the training did not converge to an optimum yet, indicating further improvement possible with more epochs. Furthermore, interestingly, scheduled slopes were able to achieve near-optimal training performance in all training rounds. Therefore, this approach is beneficial for training robustness, reducing the number of hyperparameters which need to be carefully tuned.

When looking at online reinforcement learning settings, advantage can be taken from the increased noise of shallower surrogate gradient slopes. As hinted in Figure 1c, the shallow surrogate gradients show to deviate from the true gradient. In RL, exploration is performed by injecting randomness in actions, or injecting parameter noise [26] during the forward pass. While these are explicit forms of exploration and will lead to more states to be visited, the effect of parameter noise on the cosine similarities of the computed gradient, and the gradient when no noise would be injected show similar trends as surrogate gradients. In online reinforcement learning setting, where no additional noise injection mechanisms for exploration are at work, the shallow surrogate gradient outperforms the steeper gradient significantly, Figure 2b.

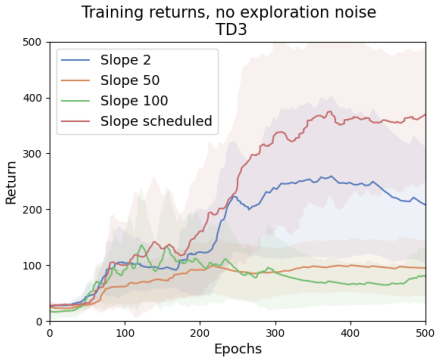
The learning signal in RL is an approximation of the true signal required for maximizing the expected returns, generated by the critic. As long as the critic can not accurately predict future returns, this signal is thus a noisy estimate of the true error signal. Therefore, a steep surrogate slope, which performs updates more true to the incoming error signal, does not necessarily lead to improved training performance. Initial experiments showed that scheduling the surrogate gradient to become steeper with improved critic accuracy, led to a significant improvement in training performance, when the schedule is tuned right. However, wrongly scheduling this surrogate gradient tended to obstruct training completely. Due to the already large hyperparameter space in RL, it was decided to use fixed surrogate gradients for all subsequent experiments.

4.2 Online Reinforcement Learning for Training SNN

Now that an understanding of the effect of the slope parametrizing the surrogate gradient has been established, different approaches to training SNNs on temporal information are compared.



(a) Noise is injected during training to increase exploration. Steep slopes only update a limited number of weights, leading to inefficient training, when the error signal itself is noisy.



(b) No additional noise is injected during training. The increased noise in gradients of the shallow gradient allows for effective exploration. Scheduling the gradient prevents converging to local minima and reduces gradient noise during training.

Figure 2: The SNNs receive full action history. This enables leveraging standard reinforcement learning frameworks, which learn based on single transitions rather than sequences.

4.2.1 Twin Delayed DDPG: TD3

Initially, it was attempted to train a network using TD3 [10], where observations are sampled sequentially from the buffer. However, as a randomly initialized network fails to generate sequences longer than the warm-up period of 50 timesteps, no gradient could be computed. Next, this period was disregarded during training. This prevents the hidden states from converging to realistic values, hindering learning for tasks where the episode length is determined by the agent's performance. The continuously changing observations combined with the unconverged hidden states decreases the validity of the learning signal significantly. The wall-clock time exceeded $24h$ before any sign of training could be observed. Furthermore, the large number of interactions required before the baseline of 50 timesteps could be achieved, called for alternative approaches.

In a second experiment, the SNN was given each observation for 4 forward passes, accumulating the spikes before performing a population-based decoding, after which the network resets its hidden states. While this is undesired, for energy efficiency and breaks the temporal relation between subsequent observations, the performance of the network is not negatively influenced by a warm-up period. This approach is common in RL with stateful neural networks, as it allows for leveraging traditional RL frameworks. This network learns much faster, but suffers from increased computational requirements, and requires manual engineering of features to create an MDP from the temporal information.

4.2.2 TD3+BC Jump-Started with a Guide Policy

For subsequent experiments, the fast reward function will not be used, as the agents trained by these experiments will be deployed on the Crazyflie. Now, an actor which has can successfully hover the Crazyflie with observations available in the simulator, is used as a jump start for bridging the warm-up period, to assure lengthy transitions to be gathered. Next, this agent is also used to gather a dataset for the BC and TD3+BC experiments. To emphasize the ability of RL to surpass the original agent, a reward curriculum is implemented.

In BC, no reward information is used, which leads to copying the guide policy. A strong decrease in performance can be seen when the reward function is altered with the curriculum. Naturally, this setup is unable to improve over the performance of the guiding agent. TD3+BC uses a critic which learns the reward structure, which in turn guides the actor. Interestingly, the presence of the critic allows the spiking actor to far exceed the BC approach in both the progressive and strict reward curriculum, as seen in Table 2. Importantly, these two algorithms have been trained on static datasets, consisting of roughly $1h30m$ of real world flight time, or 500000 environment interactions. When using an existing agent during the warm-up period, and gathering new environment interactions online, it was observed that the spiking actor was able to achieve the same performance as the BC setup. As the

Difficulty	Slope	BC	TD3+BC	TD3+BC+JSRL
Easy	2	292.0 ± 32.5	310.0 ± 84.9	353.7 ± 38.7
	100	309.7 ± 15.9	272.6 ± 5.7	309.6 ± 49.6
Medium	2	9.3 ± 23.4	123.3 ± 13.4	318.4 ± 40.7
	100	15.8 ± 28.3	124.4 ± 15.6	272.8 ± 86.0
Hard	2	-266.5 ± 140.2	-79.5 ± 3.4	90.6 ± 15.5
	100	-312.0 ± 92.7	-140.6 ± 19.3	89.0 ± 30.1

Table 2: Training SNN using 3 different algorithms. The original agent, used in BC, TD3+BC and as jump-start in TD3+BC+JSRL, was trained on the original reward function, after which curriculum training is used to improve performance of the final agents. The difficulty levels relate to the difficulty of the reward function, where easy relates to the original reward function, medium to a progressive reward function, and hard a strict reward function, presented in subsection A.2. Green cells represent the best performing algorithm, orange cells represent the second-best algorithm.

original actor, cloned in the BC setup, was trained using single transitions, using additional states, it can be concluded that online recurrent RL which is adapted to overcome the warm-up period with JSRL, can successfully achieve similar performance. Especially when utilizing a curriculum reward function, the advantages of the novel interactions shine through. The jump-started RL method was able to outperform both BC and TD3+BC.

As discussed earlier, one can either use an expert agent to jump-start training by rolling in the spiking policy throughout training, or use an agent only to bridge the warm-up period. The advantage of the aforementioned option, is increased learning speed, thanks to the BC term. This approach is similar to offline RL, as most of the training is achieved early on in training, where the guiding agent still gathers a large portion of the full rollout. While advantageous with a guiding agent which achieves high reward scores, this also leads to a final policy which displays similar behavior as the guide. Therefore, the guiding policy should only be used for the warm-up period when it is desired that the spiking policy displays different behavior.

4.3 Bridging the Reality Gap

When deploying the trained agents on the CrazyFlie, one has to take into account the computational constraints of the hardware. Where large agents or agents that perform multiple forward passes per observation can perfectly be executed in simulation, in real life, the controller needs to be running at a frequency of at least 100Hz, on an edge device. The inability to achieve this control frequency obstructs the agent to control the drone. Using NeuroBench [40], the computational requirements of the ANN, SNN which performs multiple forward passes per prediction, and finally the SNN trained on sequences can be compared.

As shown in Table 3, the larger size of the SNN compared to the ANN leads to significantly different results. The advantages of SNN become clear when looking at metrics like the activation sparsity and the synaptic operations. As the ANN uses Tanh activation functions, the output is virtually never sparse, this leads to dense synaptic operations, which is defined by the number of operations needed for a forward pass, while neglecting any sparsity, equal to the effective synaptic operations, which are the operations where only non-zero activations or weights are being multiplied. The latter therefore reflects the computational requirements on sparsity aware hardware. Next, looking at synaptic operations, one can distinguish between multiply-accumulates (MACs) and accumulates (ACs). In practice, MACs are more computational intensive and can lead to energy consumption roughly three times as large as ACs. Therefore, one can reason that the SNN, which uses temporal information, is more energy efficient than the ANN, which is smaller. Next, interestingly, the SNN with no temporal capabilities requires the large majority of operations in the encoding layer, of size 148, to accommodate for the action history, required to successfully fly the drone, as reflected in the effective MACs. This is important to take into account when deciding on the platform to deploy the model. A fully neuromorphic platform would be optimized for the binary spikes, reflected in the ACs.

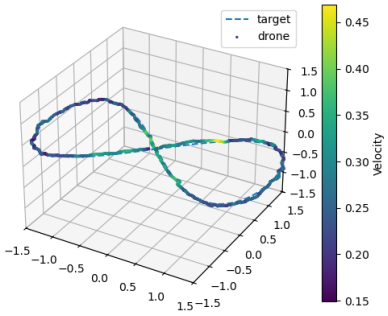


Figure 3: The spiking controller can successfully guide the drone to perform eight figures in simulation.

As most operations still happen in the non-spiking domain, to enable encoding, it would not be worth it to deploy the non-temporal SNN on neuromorphic hardware.

Baseline	ANN	$SNN_{non-temporal}$	$SNN_{temporal}$
Reward (mean $\pm$ std)	447 ± 0.9	310 ± 10.1	446 ± 1.2
Risk	445	286	442
Footprint (bytes)	55.3×10^3	287.4×10^3	158.3×10^3
Activation Sparsity	0.0	0.87	0.79
SynOps Dense	13.7×10^3	282.6×10^3	37.9×10^3
SynOps Eff_MACs	13.7×10^3	149.6×10^3	4.6×10^3
SynOps Eff_ACs	0.0	18.3×10^3	12.2×10^3

Table 3: NeuroBench results for an ANN, SNN with no temporal capabilities, and our SNN trained with TD3BC+JSRL. The model sizes with comparable performance are 64-64 for the ANN, 256-256 for the SNN with no temporal capabilities and 128-128 for the SNN with temporal capabilities.

The SNN receives the position, linear velocity, orientation and angular velocities, and outputs a motor command for each motor, in a deterministic manner. The behavior of the drone was observed to display strong oscillatory behavior. This could be explained due to the lack of penalty on angular velocities in the default reward function. Where such behavior was not observed in an ANN, which receives full action history, the SNN shows this behavior with and without the action history. As the final layer, which is further decoded, directly dictates the granularity of the outputs possible, one could reason that the network would benefit more from a reward function which penalizes abrupt behavior more than for ANN. On top of this granularity, it is observed that the throttle of existing controllers, and the networks, hover around 66%, with changes of only a few percentage points leading to abrupt changes in attitude, the network should learn to increase granularity around the hover setting. In future work, one could decide to control deviations around the drone hover throttle settings, rather than the absolute throttle setting.

Although strong oscillatory behavior is present, the spiking controller still enabled autonomous control for a wide range of maneuvers including eight figures, Figure 3.

Next, the actor is ported to the real Crazyflie. For portability reasons, the spiking policy is running on a Teensy microcontroller, which is attached to the Crazyflie. However, the Crazyflie has sufficient computational resources onboard, that the trained network can also be ran on the Crazyflie itself, at a frequency of $100Hz$. This does influence the inertia and mass of the drone, which the network will need to adapt to. Next, the network was deployed on 2 versions of the Crazyflie. One with the normal motors and propellers, as modeled in simulation, and one with upgraded motors and upgraded propellers. The latter will demonstrate the robustness of the trained network. We compare

the performance on both position control, i.e. hovering around a set point, and trajectory following. These are compared to the ANN proposed by Eschmann et. al. [6]. For fair comparison, their minimum error across all tested models is used as baseline. We compare both the proposed ANNs trained with and without action history.

While the SNN seems to be able to achieve a smaller error than the ANN, it is important to note that the spiking controller is less reliable than the ANN. Where the ANN can consistently fly the drone, the spiking actor tends to struggle to obtain similar reliability.

	Position Control Error [m]	**Trajectory Tracking Error [m]**
ANN, action history[6]	0.1	0.21
ANN, no action history[6]	0.25	∞
SNN	0.04	0.24
SNN, altered drone	0.14	0.20

Table 4: Comparison of neural network models for position control and trajectory tracking tasks. The models include an ANN with full observation and action history, an ANN without action history, an SNN trained with TD3BC+JSRL without action history, and the same SNN deployed on a drone with modified motors and propellers. The mean position error of the deployed SNNs is benchmarked against the best-performing ANN policy [6]. Position error is measured as the average xy-plane error (in meters), and trajectory tracking error is evaluated as the average error across figure-eight and square-following tasks.

5 Conclusion

Despite the widely recognized temporal processing capabilities of stateful networks like recurrent and spiking neural networks, effective training methods to harness these strengths in robotics applications remain underdeveloped. Neuromorphic computing, known for its energy efficiency and low latency, has garnered significant attention from the robotics community. In the context of spiking neural networks, the relatively recent introduction of surrogate gradient techniques has made it possible to leverage conventional deep learning frameworks, thereby facilitating the integration of various reinforcement learning algorithms.

In this work, different approaches for training spiking neural networks in robotics, ranging from supervised learning to online reinforcement learning, are examined. Additionally, a hybrid framework is proposed to accelerate the training of stateful neural networks, extending beyond just spiking neural networks. A trained agent was finally benchmarked using NeuroBench and deployed on the Crazyflie drone, demonstrating the increased efficiency of training spiking neural networks on sequential information.

When comparing behavioral cloning, offline RL and online RL, it is observed that all approaches shine in specific contexts. Where BC can clone the behavior of a controller, without the need of a reward structure, offline RL demonstrates improving over the original agent, at cost of the introduction of a reward function, which can be challenging to design. Lastly, when one wants to fully exploit a simulator, using an agent, which is only required to survive, rather than complete a task, as required in the 2 aforementioned approaches, the online RL with a jump-start demonstrates the ability of achieving similar performance to an expert controller, used in the BC and offline RL approach.

Next, the surrogate gradient parametrizing the weight update behavior of spiking neural networks, has shown to display an important role in online RL, where shallow gradients demonstrated the ability to introduce exploration. Therefore, it is advised to consider this when applying online RL to spiking neural networks.

While the spiking policy was successfully deployed to the real world, the reliability of this controller should be improved. The controller caused strong oscillatory behavior, causing some flights to terminate early due to instabilities. This could be resolved by introducing a penalty for angular velocities, and potentially modelling the effect of different propellers, motors, and the Teensy microcontroller.

REFERENCES

[1] Mahmoud Akl, Yulia Sandamirskaya, Florian Walter, and Alois Knoll. Porting deep spiking q-networks to neuromorphic chip loihi. In *International Conference on Neuromorphic Systems 2021*, pp. 1–7, 2021.

[2] Guillaume Bellec, Franz Scherr, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets, 2019. URL http://arxiv.org/abs/1901.09049https://arxiv.org/pdf/1901.09049.pdfhttps://arxiv.org/abs/1901.09049. Comment: We changed in this version 2 of the paper the name of the new learning algorithms to e-prop, corrected minor errors, added details – especially for resulting new rules for synaptic plasticity, edited the notation, and included new results for TIMIT.

[3] Zhenshan Bing, Claus Meschede, Florian Röhrbein, Kai Huang, and Alois C Knoll. A survey of robotics control based on learning-inspired spiking neural networks. *Frontiers in Neurorobotics*, 12, 2018. ISSN 1662-5218. doi: 10.3389/fnbot.2018.00035. URL https://www.frontiersin.org/articles/10.3389/fnbot.2018.00035https://www.frontiersin.org/articles/10.3389/fnbot.2018.00035/pdf.

[4] Ding Chen, Peixi Peng, Tiejun Huang, and Yonghong Tian. Deep reinforcement learning with spiking q-learning. *arXiv.org*, 2022.

[5] Sayeed Shafayet Chowdhury, Nitin Rathi, and Kaushik Roy. *Towards Ultra Low Latency Spiking Neural Networks for Vision and Sequential Tasks Using Temporal Pruning*, volume 13671, pp. 709–726. Springer Nature Switzerland, 2022. ISBN 978-3-031-20082-3 978-3-031-20083-0. URL https://link.springer.com/10.1007/978-3-031-20083-0_42https://www.ecva.net/papers/eccv_2022/papers_ECCV/papers/136710709.pdf.

[6] Jonas Eschmann, Dario Albani, and Giuseppe Loianno. Learning to fly in seconds. 11 2023. URL http://arxiv.org/abs/2311.13081.

[7] Daniel E Feldman. The spike-timing dependence of plasticity. *Neuron*, 75(4):556–571, 2012.

[8] Răzvan V Florian. A reinforcement learning algorithm for spiking neural networks. *Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 299–306, 2005. doi: 10.1109/synasc.2005.13.

[9] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. URL https://github.com/sfujim/TD3_BC.

[10] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

[11] Julia Gygax and Friedemann Zenke. Elucidating the theoretical underpinnings of surrogate gradient learning in spiking neural networks.

[12] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 1 2018. URL http://arxiv.org/abs/1801.01290.

[13] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. 2019. URL https://openreview.net/pdf?id=r1lyTjAqYX.

[14] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620:982–987, 2023. ISSN 1476-4687. doi: 10.1038/s41586-023-06419-4. URL https://www.nature.com/articles/s41586-023-06419-4https://www.nature.com/articles/s41586-023-06419-4.pdf.

[15] Richard Kempter, Wulfram Gerstner, and J Leo Van Hemmen. Hebbian learning and spiking neurons. *Physical Review E*, 59(4):4498, 1999.

[16] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019. URL https://arxiv.org/abs/1509.02971.

[17] Guisong Liu, Wenjie Deng, Xiurui Xie, Li Huang, and Huajin Tang. Human-level control through directly trained deep spiking q-networks. *IEEE transactions on cybernetics*, pp. 1–12, 2022. doi: 10.1109/tcyb.2022.3198259.

[18] Yuxiang Liu and Wei Pan. Spiking neural-networks-based data-driven control. *Electronics*, 12:310, 2023. ISSN 2079-9292. doi: 10.3390/electronics12020310. URL https://www.mdpi.com/2079-9292/12/2/310https://www.mdpi.com/2079-9292/12/2/310/pdf?version=1673945908.

[19] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. 10:1659–1671, 1997.

[20] Thomas Miconi. Biologically plausible learning in recurrent neural networks reproduces neural dynamics observed during cognitive tasks. *eLife*, 6:e20899, 2017. ISSN 2050-084X. doi: 10.7554/eLife.20899. URL https://doi.org/10.7554/eLife.20899https://elifesciences.org/download/aHR0cHM6Ly9jZG4uZWxpZmVzY2llbmNlcy5vcmcvYXJ0aWNsZXMvMjA4OTkvZWxpZmUtMjA4OTktdjIuc
elife-20899-v2.pdf?_hash=W0aFhuHQVJNohAxQoL1eTVP%2FdM48mukaN%2F4B2YaDiFc%3D.

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. URL https://arxiv.org/abs/1312.5602.

[22] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016. URL http://arxiv.org/abs/1602.01783https://arxiv.org/pdf/1602.01783v2.pdfhttps://arxiv.org/abs/1602.01783.

[23] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks.

[24] Federico Paredes-Vallés, Jesse Hagenaars, Julien Dupeyroux, Stein Stroobants, Yingfu Xu, and Guido de Croon. Fully neuromorphic vision and control for autonomous drone flight, 2023. URL http://arxiv.org/abs/2303.08778https://arxiv.org/pdf/2303.08778.pdfhttps://arxiv.org/abs/2303.08778.

[25] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. URL www.goo.gl/b57WTs.

[26] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

[27] Samuel Schmidgall and Joe Hays. Meta-spikepropamine: learning to learn with synaptic plasticity in spiking neural networks. *Frontiers in neuroscience*, 17, 2023. ISSN 1662-4548. doi: 10.3389/FNINS.2023.1183321. URL https://pubmed.ncbi.nlm.nih.gov/37250397/.

[28] Samuel Schmidgall, Jascha Achterberg, Thomas Miconi, Louis Kirsch, Rojin Ziaei, S Pardis Hajiseyedrazi, and Jason Eshraghian. Brain-inspired learning in artificial neural networks: a review.

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[30] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018. URL http://arxiv.org/abs/1506.02438https://arxiv.org/pdf/1506.02438.pdf.

[31] David Silver, Satinder Singh, Doina Precup, and Richard S. Sutton. Reward is enough, 10 2021. ISSN 00043702.

[32] Stein Stroobants, Julien Dupeyroux, and Guido De Croon. Design and implementation of a parsimonious neuromorphic pid for onboard altitude control for mavs using neuromorphic processors. In *ICONS: International Conference on Neuromorphic Systems*, pp. 1–7. ACM, 2022. ISBN 978-1-4503-9789-6. doi: 10.1145/3546790.3546799. URL https://dl.acm.org/doi/10.1145/3546790.3546799https://dl.acm.org/doi/pdf/10.1145/3546790.3546799.

[33] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[34] Guangzhi Tang, Neelesh Kumar, and Konstantinos P. Michmizos. Reinforcement co-learning of deep and spiking neural networks for energy-efficient mapless navigation with neuromorphic hardware. 3 2020.

[35] Guangzhi Tang, Neelesh Kumar, Raymond Yoo, and Konstantinos P. Michmizos. Deep reinforcement learning with population-coded spiking neural network for continuous control. 10 2020. URL http://arxiv.org/abs/2010.09635.

[36] Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, Sergey Levine, and Karol Hausman. Jump-start reinforcement learning. 4 2022. URL http://arxiv.org/abs/2204.02372.

[37] Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwi´nskabarwi´nska, Jack Rae, Piotr Mirowski, Joel Z Leibo, Adam Santoro, Mevlana Gemici, Malcolm Reynolds, Tim Harley, Josh Abramson, Shakir Mohamed, Danilo Rezende, David Saxton, Adam Cain, Chloe Hillier, David Silver, Koray Kavukcuoglu, Matt Botvinick, Demis Hassabis, and Timothy Lillicrap. Unsupervised predictive memory in a goal-directed agent. 2018.

[38] Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Recurrent policy gradients.

[39] Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896): 223–228, 2022.

[40] Jason Yik, Korneel Van den Berghe, Douwe den Blanken, Younes Bouhadjar, Maxime Fabre, Paul Hueber, Denis Kleyko, Noah Pacik-Nelson, Pao-Sheng Vincent Sun, Guangzhi Tang, et al. Neurobench: A framework for benchmarking neuromorphic computing algorithms and systems. *arXiv preprint arXiv:2304.04640*, 2023.

[41] Luca Zanatta, Francesco Barchi, Andrea Bartolini, and Andrea Acquaviva. Artificial versus spiking neural networks for reinforcement learning in uav obstacle avoidance. *ACM International Conference on Computing Frontiers*, 2022. doi: 10.1145/3528416.3530865.

A Appendix

A.1 Effect of Surrogate Gradients on Weight Updates

The choice of surrogate gradient slope in a spiking neural network is an important consideration, especially for deeper neural networks. When analyzing the effect of the slope on a single layer network, the effects are obvious. Choosing a steeper slope results in increasing the range of inputs to the neuron for which a gradient exists becomes larger, at the cost of introducing a bias in the weight update. However, when applying surrogate gradients in a deeper network, the effect becomes more complex.

Let's start by considering a network with two hidden layers, where the output layer is a weighted sum of the last hidden layer, which corresponds to population encoding. Replacing the heaviside step function in the LIF neuron model by a sigmoid and assuming β as defined in subsection 3.2 is 0, we arrive at the following neural network.

1. **Input layer to hidden layer:**

Let $\mathbf{x} \in \mathbb{R}^n$ be the input vector, $\mathbf{W_1} \in \mathbb{R}^{h \times n}$ be the weight matrix between the input and hidden layer, $\mathbf{b_1} \in \mathbb{R}^h$ be the bias vector, and $\sigma(z) = \frac{1}{1+e^{-z}}$ be the sigmoid activation function.

$$\mathbf{z_1} = \mathbf{W_1 x} + \mathbf{b_1}$$
$$\mathbf{a_1} = \sigma(\mathbf{z_1})$$

2. **Hidden layer to hidden layer:** Let $\mathbf{W_3} \in \mathbb{R}^{h \times h}$ be the weight matrix between the first hidden layer and the second hidden layer, $\mathbf{b_2} \in \mathbb{R}^h$ be the bias vector.

$$\mathbf{z_2} = \mathbf{W_2 a_1} + \mathbf{b_2}$$
$$\mathbf{a_2} = \sigma(\mathbf{z_2})$$

3. **Hidden layer to output layer:**

Let $\mathbf{W_3} \in \mathbb{R}^{o \times h}$ be the weight matrix between the hidden layer and the output layer, $\mathbf{b_3} \in \mathbb{R}^o$ be the bias vector.

$$\mathbf{z_3} = \mathbf{W_3 a_2} + \mathbf{b_3}$$
$$\mathbf{a_3} = \mathbf{z_3}$$

Thus, the final output of the 2 hidden layers neural network is $\mathbf{a_3}$.

For each neuron i in layer l, the desired change in input can be calculated as:

$$\delta_i^l = (\sum_{j=1}^{(n_{l+1})} W_{ij}^{(l+1)} \delta_j^{(l+1)})(\sigma'(z_i^{(l)}))$$

where

$$\delta_i^{(3)} = \frac{\partial L}{\partial a_i^{(3)}}$$

Where $n_{(l)}$ is defined as the number of neurons in layer l. The weight update can now be calculated as the input to the neuron multiplied by its desired output change δ:

$$\frac{\partial L}{\partial W_{ji}^{(l)}} = a_j^{(l-1)} \delta_i^{(l)}$$

Using a surrogate gradient means replacing $\sigma'(z^{(l)})$ by a surrogate function $\sigma'_k(z^{(l)})$, which is a slope-altered sigmoid parametrized by k. Popular surrogate functions in SNN include the sigmoid, fast-sigmoid and arctan. For this example, a bias-corrected sigmoid derivative will be used.

$$\sigma'_k(x) = \frac{\sigma'(kx)}{k^2} = \sigma(kx) \cdot (1 - \sigma(kx))$$

For weights in the second weight matrix, the partial derivative for $W_{ji}^{(2)}$ to L is computed as:

$$\frac{\partial L}{\partial W_{ji}^{(2)}} = a_j^{(1)}\delta_i^{(2)} = a_j^{(1)}(\sum_{j=1}^{(n_3)} W_{ij}^{(3)}\delta_j^{(3)})(\sigma'(z_i^{(2)}))$$

where replacing the activation function in the backward pass with the surrogate:

$$\frac{\partial \tilde{L}}{\partial W_{ji}^{(2)}} = a_j^{(1)}\tilde{\delta_i^{(3)}} = a_j^{(1)}(\sum_{j=1}^{(n_3)} W_{ij}^{(3)}\delta_j^{(3)})(\sigma_k'(z_i^{(2)}))$$

Assume that all weights and bias matrices, the inputs and the error, $\frac{\partial L}{\partial a_i^{(3)}}$, follow independent distributions:

$$W^{(l)} \sim \mathcal{N}(\mu_W^{(l)}, \gamma_W^{(l)})$$
$$b^{(l)} \sim \mathcal{N}(\mu_b^{(l)}, \gamma_b^{(l)})$$
$$x \sim \mathcal{N}(\mu_{in}, \gamma_{in})$$
$$\frac{\partial L}{\partial a_i^{(3)}} \sim \mathcal{N}(\mu_e, \gamma_e)$$

Then the bias found by using the surrogate gradient over the true gradient can be calculated.

$$bias = \mathbb{E}[\frac{\partial \tilde{L}}{\partial W_{ji}^{(2)}}] - \mathbb{E}[\frac{\partial L}{\partial W_{ji}^{(2)}}]$$
$$= \mathbb{E}[a_j^{(1)}] \cdot \mathbb{E}[\tilde{\delta_i^{(3)}} - \delta_i^{(3)}]$$
$$= \mathbb{E}[a_j^{(1)}] \cdot \mathbb{E}[\sum_{j=1}^{(n_3)} W_{ij}^{(3)}\delta_j^{(3)}] \cdot \mathbb{E}[\sigma_k'(z_i^{(2)}) - \sigma'(z_i^{(2)})]$$

Therefore:

$$bias = 0$$
$$\iff \mathbb{E}[\sigma_k'(z_i^{(2)})] = \mathbb{E}[\sigma'(z_i^{(2)})]$$

This can only be true if k approaches one. Note that

$$\forall x \in [-\infty, \infty] : \sigma_k'(x) >= \sigma'(x)$$

Therefore, a positive bias is introduced even in the first layer behind the surrogate gradient, which scales with $1/k$.

However, when deeper networks are considered, sign reversal of the gradient can occur [11], which perturbs the gradient approximation of deeper layers. Looking at the gradients found in $W^{(1)}$

$$\frac{\partial \tilde{L}}{\partial W_{ji}^{(1)}} = x_j\delta_i^{(2)} = x_j[\sum_{j=1}^{(n_2)} W_{ij}^{(2)}(\sum_{h=1}^{(n_3)} W_{jh}^{(3)}\delta_h^{(3)})(\sigma_k'(z_j^{(2)}))](\sigma_k'(z_i^{(1)}))$$

And realizing the cosine similarity can only equal 1 when:

$$\frac{\frac{\partial \tilde{L}}{\partial W_{ji}^{(1)}}}{\frac{\partial L}{\partial W_{ji}^{(1)}}} = 1$$

We find that this happens when:

$$\frac{x_j[\sum_{j=1}^{(n_2)} W_{ij}^{(2)}(\sum_{h=1}^{(n_3)} W_{jh}^{(3)}\delta_h^{(3)})(\sigma_k'(z_j^{(2)}))](\sigma_k'(z_i^{(1)}))}{x_j[\sum_{j=1}^{(n_2)} W_{ij}^{(2)}(\sum_{h=1}^{(n_3)} W_{jh}^{(3)}\delta_h^{(3)})(\sigma'(z_j^{(2)}))](\sigma'(z_i^{(1)}))} = 1$$

Considering the derivative of the surrogate gradient is always greater than zero, and thus greater than the true gradient, the deviation from the true gradient greatly increases with deeper layers, due to the constant overestimation.

A.2 Simulator Details

The reward structure and terminal conditions used during training greatly influence the behavior of the final policy. Picking the reward structure to be too strict, will obstruct the policy to learn at all, while soft reward structures will lead to unusable policies. In general, the right reward is crucial for successful reinforcement learning [31].

Therefore, multiple reward structures were used. The actor receives a reward for survival, C_r, avoiding the *learning to terminate* problem. Next, a penalty is given for position errors, velocity errors, orientation errors and for the magnitude of the action, parametrized with C_p, C_v, C_q and C_a respectively.

	C_p	C_v	C_a	C_q	C_r
Original	1.0	0.005	0.01	0.25	1.0
Progressive	5.0	0.05	0.3	0.25	1.0
Strict	5.0	0.2	0.5	0.25	1.0
Fast Learning	0.1	0.0	0.0	0.0	1.0

Table 5: Parameters used for reward computation.

To simulate the drone, a simple drone dynamics model is used to compute the state change after applying specific thrust to the four motors. As the model outputs an RPM setpoint, the change in RPM can be calculated using a low-pass filter.

$$\Delta rpm = (\frac{rpm_{des} - rpm_{curr})}{\tau} \tag{7}$$

Where rpm_{des} and rpm_{curr} are the desired and current rpm respectively, and τ is the time constant of the motors and propellers, reflecting the delay in actual spin up or down of the motors.

The RPM can then be converted to motor thrust using a second order model.

$$T = c_0 + c_1 \cdot rpm + c_2 \cdot rpm^2 \tag{8}$$

Where c_0, c_1, c_2 are thrust constants. The torques applied to the drone are caused by the thrust and rotor configuration.

The state changes can now be computed in the body reference frame, after which they are converted to the world reference frame.

For the parameters used, the reader is referred to the public GitHub page[1].

A.3 Recurrent Importance Sampling

In off-policy reinforcement learning (RL), the replay buffer is populated with interactions from past policies, which may differ from the current policy. A key question is whether transitions in the buffer, gathered under a different policy, can still be used to train the current policy. This issue is addressed through the concept of *importance sampling*, where we use a ratio to weigh transitions, offering insights into how relevant a particular transition is for updating the current policy.

The policy gradient can be derived by starting with the standard objective function in RL:

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\right]$$

This can be rewritten as:

$$= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau)$$

where $P(\tau|\theta)$ is the probability of observing a trajectory τ under a policy parameterized by θ.

The probability $P(\tau|\theta)$ can be expanded as:

$$P(\tau|\theta) = \rho_0(s_0)\prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t) \tag{9}$$

Here:

[1] https://github.com/korneelf1/SpikingCrazyflie

- $\rho_0(s_0)$ is the distribution of the initial state s_0,
- $P(s_{t+1}|s_t, a_t)$ is the environment transition probability from state s_t to s_{t+1} given action a_t,
- $\pi_\theta(a_t|s_t)$ is the policy function, which defines the probability of taking action a_t in state s_t.

To incorporate importance sampling, we utilize the following equation:

$$\mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\right] = \mathbb{E}_{\tau\sim\pi_{\theta'}}\left[\frac{P(\tau|\theta)}{P(\tau|\theta')}R(\tau)\right] \tag{10}$$

This equation forms the basis for off-policy RL, showing that we can estimate the gradient of the current policy using trajectories collected under a different policy, $\pi_{\theta'}$. Even in on-policy methods, this ratio is crucial, for instance, in advantage function estimations.

When applying Equation 9 to Equation 10, and accounting for recurrent policies, we get:

$$\mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\right] = \mathbb{E}_{\tau\sim\pi_{\theta'}}\left[\frac{\rho_0(s_0)\prod_{t=0}^{T} P(s_{t+1}|s_t,a_t)\pi_\theta(a_t|\left[s_t, s_{t-1}, \ldots, s_0\right])}{\rho_0(s_0)\prod_{t=0}^{T} P(s_{t+1}|s_t,a_t)\pi_{\theta'}(a_t|\left[s_t, s_{t-1}, \ldots, s_0\right])}R(\tau)\right] \tag{11}$$

This expression highlights that, for recurrent policies, the importance sampling ratio now takes into account a history of states, $[s_t, s_{t-1}, \ldots, s_0]$, instead of just the current state s_t.

In practical terms, this means that off-policy learning for recurrent or spiking neural networks does not alter the overall structure of the policy gradient update—other than introducing a sequential dependence on past states. The same principle applies: we can use transitions from other policies as long as we properly weight them according to the importance sampling ratio.

4.1. Additional Work

In the process of developing a reliable and effective training methods for SNN using RL, many approaches were experimented with. This section serves as an overview of the most important directions which were investigated.

4.1.1. Neuron Model Selection

As explained in chapter 2, various neuron models exist with their own specific dynamics, ranging from highly realistic neuron models to trimmed down, computationally efficient neuron models. Commonly, first-order LIF models are used, due to their simple implementation and limited number of parameters. A caveat of these neuron models, however, is the fact that the direct maximum effect of an incoming current occurs at the exact same time as the input itself. In contrast to second order models, where the maximum of the response towards an incoming impulse is delayed. The rise time caused by this delay can cause an outgoing spike to occur delayed towards the incoming spike, therefore exploiting the temporal dynamics of the model more.

While this is promising, and theoretically can represent more information than the first order models, it introduces another dimension to the learning problem. To compare performance, a simple model was trained to learn temporal dynamics and integrate incoming data in a supervised training manner. A dataset of the drone control task was created by deploying an existing controller and the output were motor commands. Interestingly, it was found that in this setup, the first-order models outperformed the

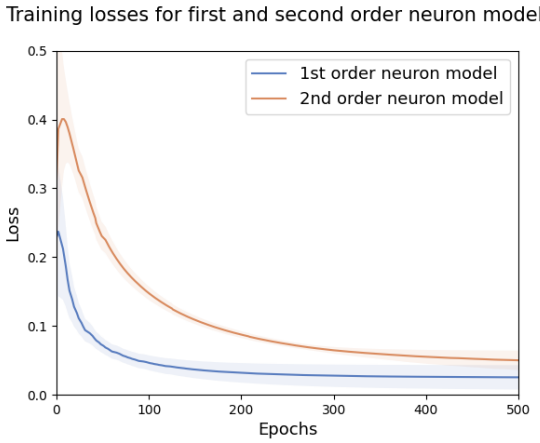


Figure 4.1: Comparison of First-Order and Second-Order Neuron Models in Spiking Neural Networks for Drone Control. The first-order neuron model demonstrates higher loss compared to the second-order neuron model. This indicates that the second-order model is better at learning the mapping from state information to motor commands, resulting in more effective control of the drone.

second-order models, see Figure 4.1. This could be explained by the fact that the drone control task already suffers from delays caused by factors such as the propeller spin-up. Additional delays could therefore obstruct successful execution of the task.

4.1.2. Parallelizable Simulator

To enable fast RL training, the speed at which one can simulate is critical. With fast simulators, one can leverage on-policy techniques, which tend to be more stable.

Initially, the FastPyDroneSim[1] was used as the dynamics model backing the RL simulation environment. This simulator provides highly optimized dynamics in Python, for drone-racing platforms. This simulator uses `numba` kernels which parallelize the operations across parallel environments and therefore enable simulation speeds that are far beyond other Python implementations available now.

A challenge with parallelized dynamics is that one can not treat each environment as independent, which can be done when other techniques such as multithreading are used. Standard RL data collection interacts with an environment until it terminates. With parallelized dynamics, however, one can not

[1] https://github.com/tudelft/fastPyDroneSim

easily reset environments that achieved terminal conditions. Therefore, two solutions were worked out. First, one simulates each environment for a fixed number of timesteps, saves the collected data, and strips the data collected after terminal conditions. While this does work, it leaves the agent to perform many steps, which are not used, wasting compute and time. However, one can place the loop which runs the simulation steps within the `numba` optimized function. A second solution would be to mask steps which occur after terminal conditions are obtained, and reset the environments in place. However, this requires checking whether terminal conditions occur during the dynamics simulation, which in turn requires step by step simulation. This is not possible within a `numba` function, exiting and entering this function introduces an overhead, which again slows down simulation. The optimal solution therefore depends on the task at hand and the hardware available. When the task terminates due to bad agent performance, which is the case in drone control tasks, the second solution, where one detects terminal conditions during simulation, can prevent numerous useless simulation steps. However, for tasks where terminal conditions are largely governed by time limits, one can confidently run the simulation for this number of steps and run the simulation loop within the `numba` function.

Upon further experimentation, it was discovered that this simulator was not optimized for the simulation of the CrazyFlie drone. This introduced a number of challenges for the RL agent, and uncertainty on whether the agent would be able to bridge the reality gap. Therefore, the usage of this optimized simulator was discarded for a slower, but CrazyFlie specific simulator. In turn, the main bottleneck of fast RL training now was the simulator. Therefore, the focus shifted towards off-policy algorithms.

4.1.3. Soft Actor-Critic

SAC is often considered as the most reliable off-policy RL algorithm. As described in chapter 3, it builds on ideas of preceding algorithms, importantly adding an entropy term to introduce improved exploration and improve training performance. Due to the attractive properties and the fact that the simulator used for training was too slow to effectively exploit the stable characteristics of on-policy algorithms such as PPO, SAC was initially used in all experiments. It was found that for ANN, SAC outperformed alternatives such as TD3 and DDPG, hinting towards the effectiveness of the entropy term. However, when applying to SNNs, it was found that the entropy term had a degrading effect. For SNNs, more basic algorithms such as TD3 or DDPG showed improved results over the entropy regulated counterpart, SAC.

One hypothesis for this behavior can be derived from chapter 4. Due to the usage of surrogate gradients, the parameter update in itself is already a noisy approximation of the gradient direction computed by the RL framework. The additional entropy term could potentially significantly decrease the training performance in SNNs. Furthermore, the end-to-end control of MAVs is an unstable environment. The simulator injects random disturbances to the MAV which already increases the exploration. Therefore, SAC was disregarded for further experiments.

4.1.4. Evolutionary Learning

Early on, it was recognized that pure online RL suffered from a major issue. In unstable environments such as drone control environments, the length of a rollout is directly affected by the performance of the agent. One major challenge is to collect rollouts which are long enough to accurately represent the temporal characteristics of, eg. successful drone flight. Especially when calculating gradients, this tends to cause a major issue, as backpropagation through time (BPTT) is used. Evolutionary Strategies (ES) are an alternative class of learning algorithms, which rather than optimizing a single agent through gradient calculation, leverages a large population of agents with random parameters and mutates the best performing agents to create children with improved performance. The advantage of these techniques is that it does not suffer from the issue that BPTT can be a noisy gradient calculation for sequences which are too short to reflect true temporal information. In early experiments, it was found that these techniques were able to achieve a baseline performance which enables sequence generation of sufficient length, as seen on Figure 4.2. Previous work has proposed hybrid methods of RL and ES [54]. However, in early experiments, it was found that the ES agents consistently converged to suboptimal performance, and that the initialization of an RL training round with this agent did not lead to high-performing agents. This could be explained due to the fact that the ES agents arrive in local minima, which do not necessarily serve as a useful initialization for the RL framework.

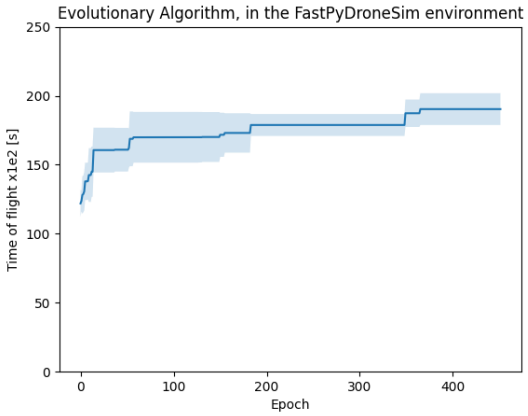


Figure 4.2: Evolutionary strategies (CEM-RL[54]) rapidly converge to a reasonable but suboptimal performance.

4.1.5. Open-Source Code

All the code used in this thesis project has been made open-source[2]. for the community to use and improve. It includes a wide variety of algorithms, simulators, and more. All files are preloaded with the hyperparameters employed in this thesis. Among these resources is the Spiking Crazyflie Gym, a high-performance simulator designed to train end-to-end control algorithms for the Crazyflie 2.1 drone. This simulator provides a robust environment for developing and testing control strategies, with a particular focus on neural network-based approaches.

In addition to the simulator, the repository includes training scripts for both artificial neural networks (ANNs) and spiking neural networks (SNNs). These scripts streamline the training process, enabling the development of effective control strategies tailored to the Crazyflie 2.1 drone.

Simulator

The simulator builds upon the Learning to Fly in Seconds project, with dynamics integrated into the Gymnasium API. The core implementation resides in the `l2f_gym.py` file, which provides the necessary interfaces and dynamics to simulate the Crazyflie 2.1 drone. Notably, as of November 2024, the Learning to Fly package supports CUDA operations, significantly improving simulation performance.

Training Agents

The repository supports multiple training methodologies, categorized into Online RL, Online-Offline RL, and Offline RL.

Online RL

Fully online RL methods are implemented in scripts such as `tianshou_l2f_<method>.py`, built upon the Tianshou RL framework. Initial experiments utilized SAC for network training, but it was found that entropy-based exploration negatively impacted performance. This was due to the environment's inherent instability and random disturbances, which already promoted sufficient exploration. For SNN training, it was observed that a shallower surrogate gradient slope improved training speed. A steep gradient slope, while precise, updates fewer weights per step, leading to slower convergence during the uncertain initial phases of RL training.

Online-Offline RL

Given that controllers for the Crazyflie 2.1 are readily available, an online-offline approach was explored to accelerate SNN training. The `TD3BC_Online.py` file implements a Jump-Start RL method using TD3+BC. This approach allows training to begin with either pre-existing replay buffers or an empty buffer initialized by an existing controller. During training, the agent initially relies on the existing controller and gradually transitions to the SNN. Mechanisms are incorporated to mitigate the impact of poor-quality data due to early terminations.

[2] `https://github.com/korneelf1/SpikingCrazyflie/`

Offline RL
Offline RL methods focus on leveraging pre-existing datasets for training. At its core, this involves supervised learning, as demonstrated in `BC.py`. For more advanced setups, actor-critic architectures can utilize reward signals to refine performance over the baseline dataset, especially when dealing with non-expert data or limited reward curricula. The `TD3BC.py` file provides an implementation of this approach, showcasing how reward information can enhance offline RL training.

This repository, in its entirety, serves as a comprehensive toolkit for developing neural network-based control strategies for drones, bridging the gap between simulation and real-world applications.

5

Conclusion

This thesis explored the integration of reinforcement learning with spiking neural networks in order to address the computational efficiency and energy constraints of real-world robotic applications. By leveraging the unique temporal capabilities of SNNs and combining them with advanced RL algorithms, a novel framework was proposed and successfully applied to drone control tasks.

The strengths and challenges of using spiking networks in an RL context were pointed out. The proposed approach of TD3+BC+JSRL demonstrated that offline and online RL can balance optimality and computational efficiency. Its asymmetric actor-critic configuration-a spiking actor combined with a non-spiking critic-simultaneously stabilizes training while offering the potential for energy-efficient deployment. The experimental results justified this, and reliable flight performance of the trained controllers on Crazyflie drones both in simulated and real environments was attained.

Despite these successes, a number of challenges remain: direct SNN training using RL methods requires a careful surrogate gradient setting with a trade-off between stability and exploration. While energy efficiency is clearly improved in the models, the decrease in reliability compared to their non-spiking equivalents calls for further investigation. Important future work remains in improving robustness in spiking networks, as well as optimizing model deployment on neuromorphic hardware.

With this thesis, the field of neuromorphic computing is advanced with further use of SNNs within RL settings. The results establish the foundation for further investigations toward low-power and time-aware learning machines standing at the crossroads of biological inspiration and engineering innovation. The encouraging results obtained in this work allow further steps toward broader adoption of neuromorphic methods in autonomous robotics and beyond.

References

[1] Mahmoud Akl et al. “Porting Deep Spiking Q-Networks to neuromorphic chip Loihi”. In: *ACM International Conference Proceeding Series* (July 2021). DOI: 10.1145/3477145.3477159. URL: https://dl.acm.org/doi/10.1145/3477145.3477159.

[2] Daniel Auge et al. “A Survey of Encoding Techniques for Signal Processing in Spiking Neural Networks”. In: *Neural Processing Letters* 53.6 (Dec. 2021), pp. 4693–4710. ISSN: 1573773X. DOI: 10.1007/S11063-021-10562-2/TABLES/1. URL: https://link.springer.com/article/10.1007/s11063-021-10562-2.

[3] John Backus. “Can programming be liberated from the von Neumann style?” In: *Communications of the ACM* 21.8 (Aug. 1978), pp. 613–641. ISSN: 15577317. DOI: 10.1145/359576.359579. URL: https://dl.acm.org/doi/10.1145/359576.359579.

[4] Guillaume Bellec et al. “A solution to the learning dilemma for recurrent networks of spiking neurons”. In: *bioRxiv* 2019 (2019), p. 738385. DOI: 10.1101/738385.

[5] Guillaume Bellec et al. *Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets*. 2019. URL: http://arxiv.org/abs/1901.09049%20https://arxiv.org/pdf/1901.09049.pdf%20https://arxiv.org/abs/1901.09049.

[6] Zhenshan Bing et al. “A Survey of Robotics Control Based on Learning-Inspired Spiking Neural Networks”. English. In: *Frontiers in Neurorobotics* 12 (2018). ISSN: 1662-5218. DOI: 10.3389/fnbot.2018.00035. URL: https://www.frontiersin.org/articles/10.3389/fnbot.2018.00035%20https://www.frontiersin.org/articles/10.3389/fnbot.2018.00035/pdf.

[7] Zhenshan Bing et al. “Indirect and direct training of spiking neural networks for end-to-end control of a lane-keeping vehicle”. In: *Neural Networks* 121 (2020), pp. 21–36. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2019.05.019. URL: https://www.sciencedirect.com/science/article/pii/S0893608019301595%20https://arxiv.org/pdf/2003.04603%20https://www.sciencedirect.com/science/article/pii/S0893608019301595.

[8] Tim Burgers, Stein Stroobants, and Guido C H E De Croon. “Evolving Spiking Neural Networks to Mimic PID Control for Autonomous Blimps”. In: ().

[9] Yongqiang Cao et al. “Spiking deep convolutional neural networks for energy-efficient object recognition”. In: *SpringerY Cao, Y Chen, D KhoslaInternational Journal of Computer Vision, 2015•Springer* 113.1 (May 2015), pp. 54–66. DOI: 10.1007/s11263-014-0788-3. URL: https://link.springer.com/article/10.1007/s11263-014-0788-3.

[10] Ding Chen et al. “Deep Reinforcement Learning with Spiking Q-learning”. In: (Jan. 2022).

[11] Ding Chen et al. “Fully Spiking Actor Network with Intra-layer Connections for Reinforcement Learning”. In: (Jan. 2024).

[12] Sérgio F. Chevtchenko and Teresa B. Ludermir. “Combining STDP and binary networks for reinforcement learning from images and sparse rewards”. In: *Neural Networks* 144 (Dec. 2021), pp. 496–506. ISSN: 0893-6080. DOI: 10.1016/J.NEUNET.2021.09.010.

[13] Iulia M. Comsa et al. “Temporal Coding in Spiking Neural Networks with Alpha Synaptic Function”. In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings* 2020-May (May 2020), pp. 8529–8533. ISSN: 15206149. DOI: 10.1109/ICASSP40776.2020.9053856.

[14] Mike Davies et al. “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning”. In: *IEEE Micro* 38.1 (2018), pp. 82–99. DOI: https://doi.org/10.1109/MM.2018.112130359.

[15] Ziyu Wang Deepmind et al. “SAMPLE EFFICIENT ACTOR-CRITIC WITH EXPERIENCE REPLAY”. In: ().

[16] PU Diehl et al. “Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing”. In: *ieeexplore.ieee.orgPU Diehl, D Neil, J Binas, M Cook, SC Liu, M Pfeiffer2015 International joint conference on neural networks (IJCNN), 2015•ieeexplore.ieee.org* (). URL: https://ieeexplore.ieee.org/abstract/document/7280696/.

[17] Jason K Eshraghian et al. “TRAINING SPIKING NEURAL NETWORKS USING LESSONS FROM DEEP LEARNING”. In: *TRAINING SPIKING NEURAL NETWORKS USING LESSONS FROM DEEP LEARNING* (2023). URL: https://snntorch.readthedocs.io/en/latest/tutorials/index.html..

[18] Răzvan V. Florian. “A reinforcement learning algorithm for spiking neural networks”. In: *Proceedings - Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2005* 2005 (2005), pp. 299–306. DOI: 10.1109/SYNASC.2005.13.

[19] Nicolas Frémaux and Wulfram Gerstner. “Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules”. In: *Frontiers in Neural Circuits* 9.JAN2016 (Jan. 2015), p. 155830. ISSN: 16625110. DOI: 10.3389/FNCIR.2015.00085/BIBTEX.

[20] Scott Fujimoto, Herke van Hoof, and David Meger. “Addressing Function Approximation Error in Actor-Critic Methods”. In: (Feb. 2018).

[21] Scott Fujimoto and Shixiang Shane Gu. *A Minimalist Approach to Offline Reinforcement Learning*. Tech. rep. URL: https://github.com/sfujim/TD3_BC.

[22] Guillermo Gallego et al. “Event-Based Vision: A Survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.1 (2022), pp. 154–180. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2020.3008413. URL: https://ieeexplore.ieee.org/ielx7/34/9639876/09138762.pdf?tp=&arnumber=9138762&isnumber=9639876&ref=%20https://ieeexplore.ieee.org/document/9138762/?arnumber=9138762.

[23] Wulfram Gerstner et al. “Eligibility Traces and Plasticity on Behavioral Time Scales: Experimental Support of NeoHebbian Three-Factor Learning Rules”. In: *Frontiers in Neural Circuits* 12 (July 2018), p. 350307. ISSN: 16625110. DOI: 10.3389/FNCIR.2018.00053/BIBTEX.

[24] Shixiang Gu et al. “Interpolated Policy Gradient: Merging On-Policy and Off-Policy Gradient Estimation for Deep Reinforcement Learning”. In: ().

[25] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: (Jan. 2018). URL: http://arxiv.org/abs/1801.01290.

[26] Nessrine Hammami and Kim Khoa Nguyen. “On-Policy vs. Off-Policy Deep Reinforcement Learning for Resource Allocation in Open Radio Access Network”. In: *IEEE Wireless Communications and Networking Conference, WCNC* 2022-April (2022), pp. 1461–1466. ISSN: 15253511. DOI: 10.1109/WCNC51071.2022.9771605.

[27] Hado van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-learning”. In: (Sept. 2015).

[28] Matthew Hausknecht and Peter Stone. “Deep Recurrent Q-Learning for Partially Observable MDPs”. In: (2015). URL: www.aaai.org.

[29] D.O. Hebb. “The Organization of Behavior : A Neuropsychological Theory”. In: *The Organization of Behavior* (Apr. 2005). DOI: 10.4324/9781410612403. URL: https://www.taylorfrancis.com/books/mono/10.4324/9781410612403/organization-behavior-hebb.

[30] Nicolas Heess et al. “Memory-based control with recurrent neural networks”. In: ().

[31] A. L. Hodgkin and A. F. Huxley. “A quantitative description of membrane current and its application to conduction and excitation in nerve”. In: *The Journal of Physiology* 117.4 (Aug. 1952), pp. 500–544. ISSN: 14697793. DOI: 10.1113/JPHYSIOL.1952.SP004764.

[32] Elia Kaufmann et al. “Champion-level drone racing using deep reinforcement learning”. en. In: *Nature* 620.7976 (2023), pp. 982–987. ISSN: 1476-4687. DOI: 10.1038/s41586-023-06419-4. URL: https://www.nature.com/articles/s41586-023-06419-4%20https://www.nature.com/articles/s41586-023-06419-4.pdf.

[33] Youngeun Kim et al. “RATE CODING OR DIRECT CODING: WHICH ONE IS BETTER FOR ACCURATE, ROBUST, AND ENERGY-EFFICIENT SPIKING NEURAL NETWORKS?” In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings* 2022-May (2022), pp. 71–75. ISSN: 15206149. DOI: 10.1109/ICASSP43922.2022.9747906.

[34] Markus Levy. *Innatera’s Spiking Neural Processor - Brain-like architecture targets ultra-low power AI*. https://www.innatera.com/innatera-mpr-2021.pdf. 2021.

[35] Timothy P. Lillicrap et al. “Continuous control with deep reinforcement learning”. In: (Sept. 2015).

[36] Guisong Liu et al. “Human-Level Control through Directly-Trained Deep Spiking Q-Networks”. In: ().

[37] Wolfgang Maass. “Networks of spiking neurons: The third generation of neural network models”. In: *Neural Networks* 10.9 (Dec. 1997), pp. 1659–1671. ISSN: 08936080. DOI: 10.1016/S0893-6080(97)00011-7.

[38] Michelle Mahowald. “VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function”. PhD thesis. CalTech, May 1992.

[39] Ana I Maqueda et al. “Event-Based Vision Meets Deep Learning on Steering Prediction for Self-Driving Cars”. en. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018, pp. 5419–5427. ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00568. URL: https://ieeexplore.ieee.org/document/8578666/%20https://openaccess.thecvf.com/content_cvpr_2018/papers/Maqueda_Event-Based_Vision_Meets_CVPR_2018_paper.pdf.

[40] Christian Mayr, Sebastian Hoeppner, and Steve Furber. *SpiNNaker 2: A 10 Million Core Processor System for Brain Simulation and Machine Learning*. 2019.

[41] Carver Mead. “Neuromorphic Electronic Systems”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1629–1636. ISSN: 15582256. DOI: 10.1109/5.58356.

[42] J Parker Mitchell et al. “NeoN: Neuromorphic control for autonomous robotic navigation”. en. In: *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*. IEEE, 2017, pp. 136–142. ISBN: 978-1-5386-1342-9. DOI: 10.1109/IRIS.2017.8250111. URL: http://ieeexplore.ieee.org/document/8250111/%20https://www.osti.gov/servlets/purl/1423018.

[43] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: (2016).

[44] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature 2015 518:7540* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236. URL: https://www.nature.com/articles/nature14236.

[45] Elias Mueggler, Chiara Bartolozzi, and Davide Scaramuzza. “Fast event-based corner detection”. In: (2017), pp. 1–8. DOI: 10.5167/uzh-138925.

[46] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. “Surrogate Gradient Learning in Spiking Neural Networks”. In: ().

[47] Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. “Recurrent Model-Free RL Can Be a Strong Baseline for Many POMDPs”. In: *Proceedings of Machine Learning Research* 162 (Oct. 2021), pp. 16691–16723. ISSN: 26403498. URL: https://arxiv.org/abs/2110.05038v3.

[48] Brendan O’donoghue et al. “COMBINING POLICY GRADIENT AND Q-LEARNING”. In: ().

[49] Katerina Maria Oikonomou, Ioannis Kansizoglou, and Antonios Gasteratos. “A Hybrid Reinforcement Learning Approach With a Spiking Actor Network for Efficient Robotic Arm Target Reaching”. In: *IEEE Robotics and Automation Letters* 8.5 (May 2023), pp. 3007–3014. ISSN: 23773766. DOI: 10.1109/LRA.2023.3264836.

[50] Federico Paredes-Vallés et al. *Fully neuromorphic vision and control for autonomous drone flight*. 2023. URL: http://arxiv.org/abs/2303.08778%20https://arxiv.org/pdf/2303.08778.pdf%20https://arxiv.org/abs/2303.08778.

[51] Devdhar Patel et al. “Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to Atari Breakout game”. In: (2019).

[52] Robert Patton et al. “Neuromorphic Computing for Autonomous Racing”. en. In: *ICONS 2021: International Conference on Neuromorphic Systems 2021*. ACM, 2021, pp. 1–5. ISBN: 978-1-4503-8691-3. DOI: 10.1145/3477145.3477170. URL: https://dl.acm.org/doi/10.1145/3477145.3477170.

[53] JA Pérez-Carrasco et al. “Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward ConvNets”. In: *ieeexplore.ieee.orgJA Pérez-Carrasco, B Zhao, C Serrano, B Acha, T Serrano-Gotarredona, S ChenIEEE transactions on pattern analysis and machine intelligence, 2013•ieeexplore.ieee.org* (). URL: https://ieeexplore.ieee.org/abstract/document/6497055/.

[54] Alo¨ıs Alo¨ıs Pourchot and Olivier Sigaud. *CEM-RL: Combining evolutionary and gradient-based methods for policy search*. Tech. rep.

[55] Henri Rebecq, Daniel Gehrig, and Davide Scaramuzza. “ESIM: an Open Event Camera Simulator”. en. In: *Conference on Robot Learning*. PMLR, 2018, pp. 969–982. URL: https://proceedings.mlr.press/v87/rebecq18a.html%20http://proceedings.mlr.press/v87/rebecq18a/rebecq18a.pdf.

[56] Bleema Rosenfeld, Osvaldo Simeone, and Bipin Rajendran. “Learning First-to-Spike Policies for Neuromorphic Control Using Policy Gradients”. In: *IEEE Workshop on Signal Processing Advances in Wireless Communications, SPAWC* 2019-July (July 2019). DOI: 10.1109/SPAWC.2019.8815546.

[57] Bodo Rueckauer et al. “Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification”. In: *Frontiers in Neuroscience* 11 (2017). ISSN: 1662-453X. URL: https://www.frontiersin.org/articles/10.3389/fnins.2017.00682%20https://www.frontiersin.org/articles/10.3389/fnins.2017.00682/pdf.

[58] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature 1986 323:6088* 323.6088 (1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://www.nature.com/articles/323533a0.

[59] Samuel Schmidgall and Joe Hays. “Meta-SpikePropamine: learning to learn with synaptic plasticity in spiking neural networks”. In: *Frontiers in neuroscience* 17 (2023). ISSN: 1662-4548. DOI: 10.3389/FNINS.2023.1183321. URL: https://pubmed.ncbi.nlm.nih.gov/37250397/.

[60] Samuel Schmidgall and Joe Hays. “SYNAPTIC MOTOR ADAPTATION: A THREE-FACTOR LEARNING RULE FOR ADAPTIVE ROBOTIC CONTROL IN SPIKING NEURAL NETWORKS”. In: ().

[61] Samuel Schmidgall et al. “SpikePropamine: Differentiable Plasticity in Spiking Neural Networks”. In: *Frontiers in Neurorobotics* 15 (Sept. 2021), p. 629210. ISSN: 16625218. DOI: 10.3389/FNBOT.2021.629210/BIBTEX.

[62] Thorben Schoepe et al. “Closed-loop sound source localization in neuromorphic systems”. en. In: *Neuromorphic Computing and Engineering* 3.2 (2023), p. 024009. ISSN: 2634-4386. DOI: 10.1088/2634-4386/acdaba. URL: https://iopscience.iop.org/article/10.1088/2634-4386/acdaba%20https://pure.rug.nl/ws/portalfiles/portal/689382619/Schoepe_2023_Neuromorph._Comput._Eng._3_024009.pdf.

[63] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: ().

[64] John Schulman et al. “Trust Region Policy Optimization”. In: (Feb. 2015).

[65] David Silver et al. “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”. In: (Dec. 2017). URL: https://arxiv.org/abs/1712.01815v1.

[66] Sen Song, Kenneth D. Miller, and L. F. Abbott. “Competitive Hebbian learning through spike-timing-dependent synaptic plasticity”. In: *Nature Neuroscience 2000 3:9* 3.9 (Sept. 2000), pp. 919–926. ISSN: 1546-1726. DOI: 10.1038/78829. URL: https://www.nature.com/articles/nn0900_919.

[67] *Speck*. https://www.synsense.ai/products/speck/.

[68] Stein Stroobants, Julien Dupeyroux, and Guido C H E de Croon. “Neuromorphic computing for attitude estimation onboard quadrotors”. In: *Neuromorphic Computing and Engineering* 2.3 (2022), p. 034005. ISSN: 2634-4386. DOI: 10.1088/2634-4386/ac7ee0. URL: http://arxiv.org/abs/2304.08802%20https://arxiv.org/pdf/2304.08802.pdf%20https://arxiv.org/abs/2304.08802.

[69] Stein Stroobants, Julien Dupeyroux, and Guido De Croon. “Design and implementation of a parsimonious neuromorphic PID for onboard altitude control for MAVs using neuromorphic processors”. en. In: *ICONS: International Conference on Neuromorphic Systems*. ACM, 2022, pp. 1–7. ISBN: 978-1-4503-9789-6. DOI: 10.1145/3546790.3546799. URL: https://dl.acm.org/doi/10.1145/3546790.3546799%20https://dl.acm.org/doi/pdf/10.1145/3546790.3546799.

[70] Shiva Subbulakshmi Radhakrishnan et al. “A biomimetic neural encoder for spiking neural network”. In: *Nature Communications 2021 12:1* 12.1 (Apr. 2021), pp. 1–10. ISSN: 2041-1723. DOI: 10.1038/s41467-021-22332-8. URL: https://www.nature.com/articles/s41467-021-22332-8.

[71] Richard S Sutton and Andrew G Barto. “Reinforcement Learning: An Introduction Second edition, in progress”. In: ().

[72] Weihao Tan, Devdhar Patel, and Robert Kozma. “Strategy and Benchmark for Converting Deep Q-Networks to Event-Driven Spiking Neural Networks”. In: (2021). URL: www.aaai.org.

[73] Guangzhi Tang, Neelesh Kumar, and Konstantinos P Michmizos. “Reinforcement co-Learning of Deep and Spiking Neural Networks for Energy-Efficient Mapless Navigation with Neuromorphic Hardware”. In: (). URL: https://github.com/combra-lab/spiking-ddpg-mapless-navigation.

[74] Guangzhi Tang et al. “Deep Reinforcement Learning with Population-Coded Spiking Neural Network for Continuous Control”. In: (Oct. 2020). URL: http://arxiv.org/abs/2010.09635.

[75] Antonio Vitale et al. “Event-driven Vision and Control for UAVs on a Neuromorphic Chip”. en. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 103–109. ISBN: 978-1-72819-077-8. DOI: 10.1109/ICRA48506.2021.9560881. URL: https://ieeexplore.ieee.org/document/9560881/%20https://rpg.ifi.uzh.ch/docs/ICRA21_Vitale.pdf.

[76] Zhihan Yang and Hai Nguyen. “Recurrent Off-policy Baselines for Memory-based Continuous Control”. In: (Oct. 2021). URL: https://arxiv.org/abs/2110.12628v1.

[77] Luca Zanatta et al. “Artificial versus spiking neural networks for reinforcement learning in UAV obstacle avoidance”. In: *ACM International Conference on Computing Frontiers* (2022). DOI: 10.1145/3528416.3530865.

[78] Luca Zanatta et al. “Directly-trained Spiking Neural Networks for Deep Reinforcement Learning: Energy efficient implementation of event-based obstacle avoidance on a neuromorphic accelerator”. In: *Neurocomputing* 562 (Dec. 2023), p. 126885. ISSN: 0925-2312. DOI: 10.1016/J.NEUCOM.2023.126885.

[79] Duzhen Zhang et al. “Multi-Sacle Dynamic Coding Improved Spiking Actor Network for Reinforcement Learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.1 (June 2022), pp. 59–67. ISSN: 2374-3468. DOI: 10.1609/AAAI.V36I1.19879. URL: https://ojs.aaai.org/index.php/AAAI/article/view/19879.

[80] Duzhen Zhang et al. “Population-coding and Dynamic-neurons improved Spiking Actor Network for Reinforcement Learning”. In: (June 2021).

[81] Yixin Zhu et al. “Evolutionary vs imitation learning for neuromorphic control at the edge * You may also like CMOS-compatible neuromorphic devices for neuromorphic perception and computing: a review Evolutionary vs imitation learning for neuromorphic control at the edge *”. In: (2022). DOI: 10.1088/2634-4386/ac45e7. URL: https://doi.org/10.1088/2634-4386/ac45e7.

A

Appendix

Supplementary materials are provided below.

A.1. Using A2C for spiking neural networks

The following work was performed as part of the Control and Simulations project. It served as a first attempt to get familiarized with developing SNN specific RL methods. While this work is far from perfect, it served as a stepping stone to the methods developed in this thesis.

A Comparative Study of Spiking and Non-Spiking Neural Networks in Actor-Critic Reinforcement Learning

K. Van den Berghe
Technische Universiteit van Delft, 1 Kluyverweg, Delft

ABSTRACT

Within the world of autonomous micro robotics, constraints in computational resources and energy usage is a pressing issue, motivating the exploration of novel innovative algorithms to bring intelligence to these systems. Currently, the state of the art in embodied intelligence makes use of artificial neural networks. These neural networks often require intensive computations and occupy a large memory footprint. Recently, neuromorphic solutions have gained attention from the machine learning community as a possible avenue to tackle the issue of energy efficiency. More specifically, spiking neural networks have shown impressive results requiring only a fraction of the operations required for similar performance to their non-spiking alternatives. While supervised, unsupervised and even self-supervised training are widely explored, reinforcement learning using these networks has received little attention. This work aims at exploring the challenges and opportunities that arise when deploying spiking neural networks as workers in actor-critic deep reinforcement learning. The reinforcement learning algorithm chosen is the A2C algorithm, optimizing an artificial neural network and a spiking neural network, using the leaky integrate and fire neuron model. The task under consideration is the balancing of a stick on a cart, the CartPole task. While the training of the spiking agents was noisy, and performance is slightly inferior to the non-spiking variant, it is found that the computational complexity for the spiking variants can be easily decreased significantly with respect to their non-spiking alternatives. Furthermore, the spiking variants showed an improved noise robustness.

1 INTRODUCTION

Reinforcement learning (RL) has experienced significant advancements in the past years, enabling robots to achieve human like performance on complex tasks. A popular category of reinforcement learning algorithms include actor-critic reinforcement learning, which has shown pivotal in training efficiently in challenging environments. These algorithms have the potential of being more sample efficient, thanks to their parallel training. Concurrently, neuromorphic algorithms, such as spiking neural networks (SNN) have garnered attention in the field of deep learning. These bio-inspired algorithms have demonstrated utility in handling temporal data and offer high energy efficiency when run on specialized hardware. Combining the possibility to learn from experience from reinforcement learning with the energy efficient characteristics of spiking neural networks holds great opportunity in the field of small mobile robotics, where computational resources are constrained.

Current research in the field of spiking based reinforcement learning algorithms is largely based on the principle of deep Q learning (DQN). While showing impressive results, this method relies on the use of a technique called experience replay to reduce the bias during training. This experience therefore needs to be gathered before the training can start. Actor-critic reinforcement learning on the other hand, makes use of multiple workers operating in parallel, eliminating the need for experience replay. This opens doors to learning on resource constrained systems, where the overhead required for storing past experiences and training in batches of experience can be detrimental for the performance of the system.

Considering the advantages of the actor-critic algorithms and spiking neural networks, this article explores the challenges and opportunities of combining the two. Firstly, the related work in the field of spiking based reinforcement learning is discussed in section 2. Going over the general use cases where these bio-inspired algorithms shine and the implementation details of each work. Next, the methods used for this article are discussed in section 3. Then, section 4 presents the results. In section 5, the results are discussed and directions for future work are given. Lastly, the article is concluded in section 6.

2 RELATED WORK

Existing research in the field of neuromorphic RL has mainly applied one algorithm; deep Q learning. The deep Q algorithm is a well established algorithm, introduced by DeepMind. The original publication demonstrated the performance of this algorithm on multiple Atari games, displaying its versatility [1]. It is a simple and elegant algorithm, based on tabular Q-learning, that uses an artificial neural network as policy. One component, important in the performance of this

algorithm, is the experience replay [2]. During the training phase, experience from previous environment interactions is used together with the most recent experience to avoid bias in the training data and improve training characteristics. The spiking variant of this algorithm has been demonstrated in earlier work. DSQN has been applied on the Atari environments and on the Airsim environment and outperform several ANN based solutions [3, 4, 5].

The previous work on SNN based RL algorithms span a wide array of different environments, tasks and training methods. Early research already harnessed bio-inspired training algorithms such as spike-timing-dependent plasticity (STDP) [6]. STDP is commonly described as: *cells that fire together, wire together*. It is a concept derived from neuroscience, that assumes that neurons that often spike simultaneously, probably represent the same information. R.V. Florian[7] proposed an algorithm based on deep Q learning, modulating the STDP with the global reward signal, for a bio-inspired problem. The agent controls a worm that solves a localization problem based on gradient descent, receiving a positive reward if it comes closer to the source, while receiving a negative reward when moving to a region with a lower concentration. Further work shows that using reward modulated spike-timing-dependent plasticity (R-STDP) or temporal difference spike-timing-dependent plasticity (TD-STDP) allows networks to train more complex tasks such as the CartPole [8], a task where the model has to balance a stick on a cart, by only applying a force to the cart [9]. However, the model showed slow convergence and noisy, imperfect results. Another method inspired by neuroscience is eProp. This method showed to approach the performance of back-propagation through time for recurrent SNN [10].
Methods that more closely resemble the training of non-spiking networks, include shadow training and surrogate gradient training. In shadow training [11]. We train an ANN and convert it to an SNN based agent. This approach, however, has shown to produce worse performing models compared to the ANN based equivalent and to a DQN algorithm directly optimizing the SNN based agent (DSQN)[12]. This is explained by the fact that the converted SNN will always be limited by the ANN from which it is converted, while the other two methods (DQN and DSQN) train directly from the environment. Surrogate gradient training [13] overcomes the non-differentiable of spikes by modelling the step function that presents the spiking behaviour of a neuron, with a differentiable surrogate function that approaches the step function the backward pass.

3 METHODS

Many components are necessary to successfully train an agent using reinforcement learning. Reinforcement learning is notoriously tricky to train, and so are spiking neural networks. Therefore, the choice of agent architectures, loss functions, neuron models etc. are of utmost importance. First the background of the reinforcement algorithm being used will be explained. Then, the network responsible for interacting with the environment is presented. Finally, the training of the algorithm is elaborated upon.

3.1 A2C Reinforcement Learning

In 2016, DeepMind introduced the Asynchronous Advantage Actor-Critic (A3C) algorithm in their paper *Asynchronous Methods for Deep Reinforcement Learning* [14]. It is built on using multiple workers who explore the environment in parallel independently and updating a global network with their experiences, asynchronously. Before a worker interacts with its environment, it updates its internal model with the global model. Compared to the popular DQN algorithm, this network did not need experience replay, trying to reduce variance and bias by using the experience from multiple workers, and thus updates the underlying model more frequently with fresh experiences. While the A3C algorithm performed well on many tasks, it was unsure whether the asynchrony, which adds complexity, actually affected the performance positively. Next to the added complexity, A3C does not take full advantage of the possibility to perform computations on large batch sizes of the GPU. Therefore, the synchronous version of A3C has been introduced, named Advantage Actor-Critic (A2C). As the ability to perform online distributed training with these algorithms shows promising avenues for fine-tuning after deployment, we decided to pursue experimentation using these algorithms rather than the DQN based algorithms, which already showed promising results. When finetuning on device, every sample is costly to obtain. Having to wait until sufficient experience has been gathered to start training, is therefore undesirable.

3.2 CartPole task

The task to be solved in this article is the CartPole task. This task has been widely used to develop new reinforcement learning algorithms due to its simple implementation, yet challenging solution. The task consists of balancing a pole on a cart as long as possible, receiving $+1$ reward for every timestep where the pole is balanced. The actor can apply either a force to the left or to the right on the cart, while observing the position and velocity of the cart and angle and angular velocity of the pole. For all models, the environment is ran at a frequency of 20Hz.

3.3 Architecture and Neuron Model

Where the conventional reinforcement learning methods use actor-critic models based on a deep neural network using artificial neurons, the aim of this article is to demonstrate the use of spiking neural networks. Spiking neural networks use biologically inspired neuron models that simulate the behavior of neurons in the brain. In the past, many different

neuron types have been proposed, with each their specific use cases. Complex neuron models are often utilized for neuroscience research, while simpler models have shown useful in machine learning research.

To compare ANN to SNN, two models (an ANN and an SNN) with the same architecture were used. The only difference between the models is the spiking neurons in the SNN, replacing the regular activation functions. This architecture is kept small to avoid long training times, which is an issue for SNN. The input layer consists of 4 continuous neurons, there is one hidden layer, with 246 neurons, and finally, two output heads represent the actor and critic output respectively. Next to this shallow network, experiments were carried out with a slightly deeper network with a similar number of neurons. This second architecture replaces the single hidden layer with 246 neurons, with two hidden layers, each having 128 neurons.

The activation function used for the ANN is the ReLU function. In the SNN experiments were conducted with two different neuron types for the hidden layer, and a non-spiking equivalent of each respective neuron is used for the output layer. Firstly, the leaky integrate fire (LIF) is used. The LIF neuron is a first-order neuron where the input current, I_{in} directly charges the membrane potential, U. This potential energy leaks over time at a rate β, the leakage parameter. When this potential exceeds a threshold, U_{thr}, the neuron spikes and the membrane potential is reset, by subtracting the threshold value. The charging and resetting of the membrane potential can be modeled using the following equation:

$$U[t+1] = \beta U[t] + I_{in}[t+1] - R \cdot U_{thr} \tag{1}$$

Where R is 1 whenever the membrane potential exceeds the threshold and 0 otherwise. The spiking behavior can be modeled as:

$$s = \begin{cases} 1, & \text{if } U[t+1] > thr \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Alternatively, a second order model, namely a synaptic neuron model was investigated as well. In second order models, the synaptic current is charged by the input current, which subsequently charges the membrane potential, after which similar behavior as the LIF can be observed. Concretely, the following equations model these dynamics.

$$I_{syn}[t+1] = \alpha I_{syn}[t] + I_{in}[t+1] \tag{3}$$

$$U[t+1] = \beta U[t] + I_{syn}[t+1] - RU_{thr} \tag{4}$$

The spiking behavior for this neuron is the same as the LIF neuron described above. However, after initial experiments, it was found that this neuron model showed very slow convergence. After analyzing the impulse response of this neuron type, it is found that it is significantly delayed compared to the first order model, which might be a cause for the slow training.

3.4 Encoding and Decoding

To deploy spiking based models, we have to translate the continuous domain to the discrete spiking domain and back to continuous domain at the output. The way the bridging of domains is carried out has significant effects on training speed and final model performance. Several successful methods have been proposed in the past, ranging from rate coded, latency coded to population coded methods. For robotic applications, learned coding has shown effective and easy to implement[15]. The continuous inputs are connected to a first layer of spiking neurons with a simple linear layer that encodes the continuous input to input current, which is fed into the first spiking layer. The output is directly read from the membrane potential from the last layer, which is programmed to never spike. While this method can learn complicated encoding and decoding mechanisms, it is not necessarily the most computationally efficient method. Depending on the layer sizes, large linear layers might be required, for which the number of multiply-accumulates equals the product of the matrix dimensions.

3.5 Training Spiking Neural Networks

Training spiking neural networks have shown to carry several unique challenges. Firstly, due to the discrete spiking nature of the neurons, one can not compute the derivative of this activation to compute the gradient of the weights running through the network. Therefore, several different training methods have been proposed. To be able to apply a reinforcement learning algorithm without significant changes to train an SNN, we would like to have a training method based on backpropagation. To find the gradient of the non-differentiable spikes, a surrogate gradient is applied during the backward pass[13]. When training using surrogate gradients, a continuous, differentiable function is used to mimic the spiking behavior. During the forward pass, the discrete spiking function is used, but during the backward pass, the derivative surrogate gradient represents the gradient of the signal through the network. The derivative of this surrogate function is a real non-zero value at the time of spiking. Its trace reduces to zero as time moves away from the spike time, seen on Figure 1. The surrogate function used is the fast sigmoid function with a slope of 25. Increasing the slope leads to shorter traces in time.

Next to the above-mentioned challenges, the choice of loss function effects the training process significantly. For policy gradient methods, generalized advantage can improve sample efficiency and increase stability during training [16]. This loss was applied to the membrane potentials of the output neurons.

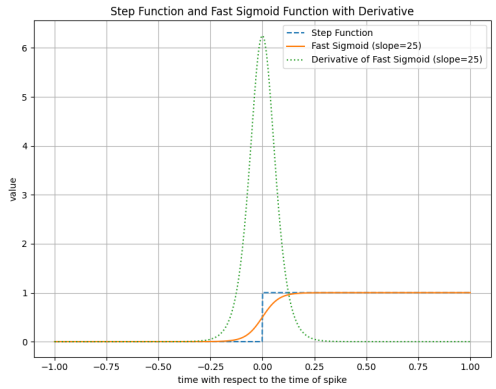


Figure 1: The step function, surrogate function and its gradient.

3.6 Pruning the Spiking Neural Network

To explore opportunities of using the trained agents on resource-constrained embedded devices, pruning and quantization are widely used to reduce the footprint and computational effort of networks. While in ANN, spatial pruning is extensively exploited, in SNN the opportunity rises to take a temporal pruning approach, with minimal performance decay. Due to undesired effects during training using RL, the trained agent often has dead or saturated spiking neurons. Dead neurons are the ones that never spike, while saturated spiking neurons spike at every timestep. This opens the opportunity to prune the network without losing any performance. Dead neurons, and the connecting weights, can be removed. Saturated spiking neurons, on the other hand, can be removed by removing the weights connecting it to the previous layer, and adding the weights connecting to the next layer as bias of the next neuron layer.

4 RESULTS

Spiking neural networks have notoriously been a challenge to train. Reinforcement learning on the other hand, is highly sensitive to factors such as hyperparameter tuning. The combination of SNNs trained with an RL framework therefore showed a challenging task. Therefore, we start by comparing the training patterns of SNNs compared to ANN. Next, an analysis is carried out on the complexity of both the neuromorphic and conventional solution, using NeuroBench. Lastly, it is analyzed how the introduction of noise on the sensor inputs affect the performance of the algorithm. This reflects the behavior of the trained agents in a more realistic setting.

4.1 Training

For the control of the CartPole, multiple policies were trained, divided in two architecture types. First, models with a single hidden layer were trained, then, models using two hidden layers were trained. Of both policy types, at least one conventional and one neuromorphic solution was trained. Due to the simplicity of this task, it can be observed that all models are able to achieve a reasonable performance. As expected, for both the deeper model as the model with only one hidden layer, the neuromorphic solution converges slower. This can partly be explained by the way the surrogate gradient, used during backpropagation, works. Where in a conventional neural network, virtually all weights contribute to the output at each timestep, for spiking neural networks this is not the case. Neurons for which the potential at a timestep is lower than the threshold, will not spike. Therefore, for this instant, there is no backpropagation gradient past this neuron.

Figure 2 shows the reward obtained by the agent after every episode of the training cycle. Allowing the network to train the amount of leakage of the neurons leads to fast convergence and relatively high performance. However, upon further inspection of the trained model, it can be seen that the model learns to have complete leakage at every timestep ($\beta = 0$ in Equation (1)). This thus corresponds to an ANN with a Heaviside threshold function which goes from zero to one at the threshold value. Therefore, no temporal dynamics are learned. In a subsequent training cycle, the leak, β was fixed to a value of 0.65. Initially this leads to faster training, as less parameters need to be learned. However, the training is less stable and the final model is not able to reach a competitive performance to the other trained models.

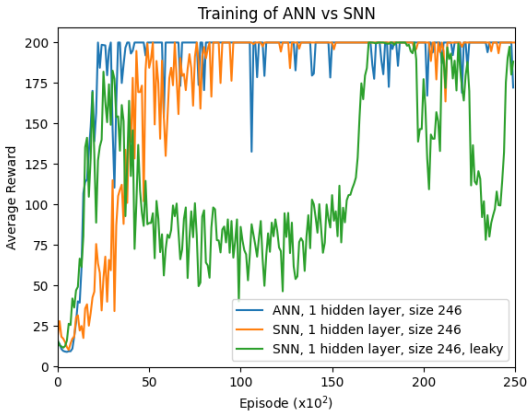


Figure 2: Training of an ANN and SNN with one hidden layer of size 246. The first SNN learns to reduce β to zero. The second SNN has a fixed leak, $\beta = 0.65$.

Next, the deeper models were trained, presented on Figure 3. These showed a slower convergence, due to the increased effect of the vanishing gradients. However, the training showed more stable behavior. Interestingly, it was found that the training of SNN showed to be easier when the weight of the entropy loss was increased, encouraging random ac-

tions. For the hyperparameters used, the public GitHub repos-

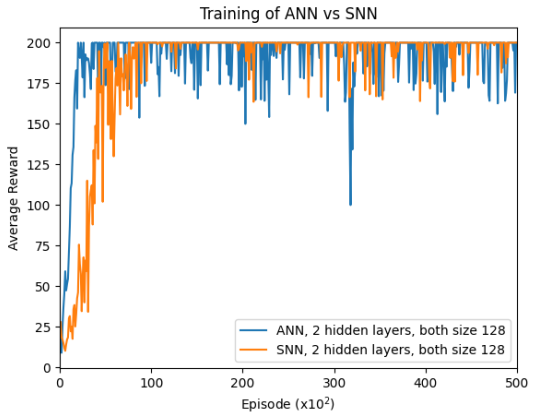


Figure 3: Training of an ANN and SNN with two hidden layers of size 128. The SNN has a leak of $\beta = 1$. $\beta = 0.95$ for the first and second spiking layer respectively.

itory can be consulted [1].

4.2 *Performance and computational complexity*

Using NeuroBench [2], a fair comparison can be made between the trained models. The benchmark provides performance and complexity metrics. For more information on the metrics, the article can be consulted. While NeuroBench does not officially support the benchmarking of closed-loop systems yet, an initial effort towards benchmarking these systems is publicly available on the GitHub repository [3]. The benchmark starts 1000 interactions with the environment, each limited to a maximum of 10000 steps and monitors the model performance and complexity. Performance for this task is defined as the average obtained reward and the standard deviation. Furthermore, inspired by A2Perf [4], the risk is computed as well. This risk is defined as the average of the 5% lowest scoring interactions. The complexity metrics on the other hand, include the footprint, model frequency (which is defined to be 20Hz for all models), the connection sparsity, activation sparsity and the synaptic operations required per prediction.

First, the shallow models, having a hidden layer of size 246 are analysed. Table 1 presents the results for the ANN, while Table 2 presents the results for the SNN without temporal dynamics, as well as the SNN with a fixed leak of $\beta = 0.65$.

As expected the ANN outperforms the SNN variants when looking at the reward characteristics. The well established training techniques and the reinforcement learning algorithm

[1] https://github.com/korneelf1/SpikingA2C

[2] https://neurobench.ai

[3] https://github.com/NeuroBench/neurobench

[4] https://github.com/Farama-Foundation/A2Perf#rlperf-benchmark-for-autonomous-agents

Baseline	ANN
Reward (mean $\pm$ std)	1744 ± 1385
Risk	190
Footprint (bytes)	7.9×10^3
Activation Sparsity	0.0
SynOps Dense	1.7×10^3
SynOps Eff_MACs	1.5×10^3
SynOps Eff_ACs	0

Table 1: NeuroBench results for the single layer ANN controlling the CartPole task.

Baseline	SNN	SNN, leaky
Reward (mean $\pm$ std)	1620 ± 1600	232 ± 65
Risk	148	152
Footprint (bytes)	7.9×10^3	7.9×10^3
Activation Sparsity	0.68	0.92
SynOps Dense	1.7×10^3	1.7×10^3
SynOps Eff_MACs	0.9×10^3	0.9×10^3
SynOps Eff_ACs	238	59

Table 2: NeuroBench results for the single layer SNNs controlling the CartPole task.

that was originally made for ANN, shows effective training of ANN for the CartPole task. Comparing the two SNN models (Figure 2, it shows that the first one, which did not learn temporal dynamics, is able to achieve a significantly higher mean reward than the leaking model. However, when looking at the standard deviation and risks of both models, the leaky SNN tends to be more predictable and even less risky.

Next, the differences in the results of the complexity metrics display where SNN models shine compared to the better performing ANN equivalent. Thanks to the spiking nature of SNN, layers after a spiking layer only require accumulates (AC) rather than multiply accumulates (MACs). Where an AC consists of one operation, the addition, the MAC requires a multiplication and an addition. The multiply operation is more energy demanding than the addition. Therefore, computing an AC is much less intensive. Furthermore, thanks to the sparsity in the spiking models, the effective operations required, is significantly reduced. These effective operations are defined as operations where non-zero inputs are multiplied by non-zero weights. Specialized hardware could make use of these characteristics and reduce the computational ef-

fort required for these operations. Furthermore, it can be seen that the encoding layer, which is a linear layer connecting the continuous input to the first spiking layer, accounts for the largest number of operations in the spiking model. Therefore, more efficient spiking encoders could significantly reduce the complexity of the model.
Table 3 shows the results on the deeper models tested through NeuroBench. It can be seen that the deeper model significantly increases the required footprint and number of dense operations. However, the number of MACs for the SNN, representing the encoding, has decreased when comparing with the single layer models with a larger hidden layer. The performance of the models is observed to decrease significantly, due to the increased effect of dead and saturated neurons in deeper models. This leads to a significant reduction in the performance of the 5% scoring interactions, reflected by the risk metric.

Baseline	ANN, 2 layers	SNN, 2 layers
Reward (mean $\pm$ std)	360 ± 116	220 ± 139
Risk	97	27
Footprint (bytes)	70×10^3	70×10^3
Activation Sparsity	0.0	0.82
SynOps Dense	17×10^3	17×10^3
SynOps Eff_MACs	10×10^3	0.5×10^3
SynOps Eff_ACs	0	4.2×10^3

Table 3: NeuroBench results for the double layer models controlling the CartPole task.

4.3 Pruning

When training spiking neural networks using surrogate gradients, two undesired effects can occur. Dead neurons and saturated neurons. This is caused by the input current being very low or very high, leading to a non-spiking or continuously spiking neuron respectively. The gradient of the surrogate function is always zero, countering the flow of gradients through the net. While this is a highly undesired behavior during training, it does leave the opportunity to get rid of these neurons. For both the single layer spiking models, these neurons were removed. Firstly, the dead neurons and their connecting synapses are completely removed. Secondly, the synapses for which the post-synaptic neuron is saturated, are removed. For the synapses for which the pre-synaptic neuron is saturated, the synaptic weight is added as a bias to the post-synaptic neuron.
To identify these neurons, the model interacts with the environment 1000 times, during which the spiking activity for each neuron is registered. Neurons with 0% or 100% spiking activity are removed as described above. This theoretically should not affect the performance, however due to neurons that have near 0% or 100% spiking activity, some neurons are wrongfully classified as dead or saturated. This effect can be reduced by increasing the number of interactions with the environment. This led to impressive results in the consequent NeuroBench tests. Running the pruning algorithm on the first single layer SNN, with a β of 0, we were able to remove 235 neurons, leaving us with a hidden layer of only 11 neurons. For the second SNN, with a β of 0.65, we were able to remove 225 neurons, which leads to a hidden layer size of 21. This is reflected in Table 5. Note that for this NeuroBench run, the critic head was removed, as this does not add any functionality during the inference of the agent. Including this head would lead to an increase in ACs of at most 11 and 21 for the pruned SNN and the pruned leaky SNN respectively.
These pruned models show the pruning possibilities when using spiking neural networks trained with A2C. While performance does not degrade significantly, the footprint and operations required reduce roughly with a factor 20 and 25 respectively. While the largest computational effort still is attributed to the encoding layer (reflected in the effective MAC metric), the reduction in complexity using non-neuromorphic hardware is still promising, now only requiring 66 dense operations, versus the 1700 operations required for the previous model.

Baseline	SNN pruned	SNN, leaky pruned
Reward (mean $\pm$ std)	1570 ± 1480	200 ± 64
Risk	140	73
Footprint (bytes)	360	640
Activation Sparsity	0.49	0.65
SynOps Dense	66	126
SynOps Eff_MACs	44	84
SynOps Eff_ACs	12	16

Table 4: NeuroBench results for the pruned single layer SNNs controlling the CartPole task.

After analyzing the great reduction in complexity after the pruning of the SNN, a model with the same architecture, replacing the spiking neurons with ReLU activation functions, was trained for the same number of episodes as the original SNN. Training this network with a single hidden layer of 11 neurons showed a significant decrease in performance comparing to the original ANN and even compared to the pruned SNNs. Where the pruned SNNs were able to have a relatively small decrease in average performance and a reasonable risk, the ANN's average performance degraded significantly with a risk that may be unacceptable in some circumstances. Again, for this analysis, the critic head was removed.

Baseline	ANN 11 neurons
Reward (mean $\pm$ std)	263 ± 114
Risk	42
Footprint (bytes)	360
Activation Sparsity	0.0
SynOps Dense	66
SynOps Eff_MACs	66
SynOps Eff_ACs	0

Table 5: NeuroBench results for the ANN with 11 hidden neurons controlling the CartPole task.

For the deeper models, it is hypothesised that similar results can be obtained. However, due to time restrictions, the pruning of deeper SNN is left as future work.

4.4 Noise robustness

When deploying controllers in the real world, the observations will be affected by noise. The noise robustness of controllers can therefore be a significant factor in the decision on what algorithm is suitable for its usecase. This subsection therefore attempts to characterize the behaviour of all aforementioned models under noise.

As explained in the previous sections, allowing the leakage to be trained causes the neuron to disregard temporal dynamics completely. Constraining the leakage to non-zero numbers, necessitates the model to learn the temporal dynamics of the system. This leads to longer and more difficult training for the SNN. On the other side, it is often hypothesized that the temporal spiking nature of SNN could lead to noise robustness. Therefore, we conducted an experiment to analyze the performance of the trained models after injecting noise, to more closely simulate real-world imperfections.

Experiments were carried out to asses the noise robustness of the trained models. For a range of a gain of 0 to 1 with a step size of 0.015, each model was evaluated 250 times, with a maximum interaction time of 1000. These parameters were chosen to sufficiently represent the behaviour under noise, while reducing the computational resources needed to generate the results.

Figure 4 displays the results of the previously described experiment. It can be seen that the models which exploit the leakage of the neurons tend to be more noise robust. At a noise level of roughly 0.15, these models outperform their non leaking counterparts. The non leaking models tend to show a high degradation of performance when noise is injected.

Performing the same analysis for the deeper models leads to similar results, as shown on Figure 5. At an injected noise with gain of 0.04, the performance of the SNN surpasses the ANN.

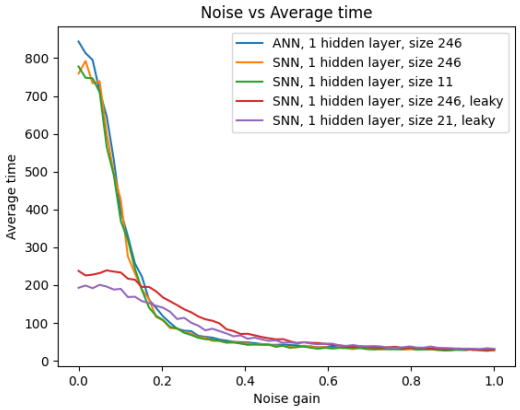


Figure 4: Noise robustness analysis of the previously described single layer models.

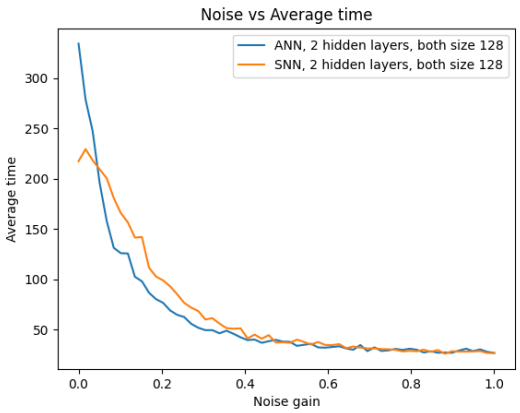


Figure 5: Noise robustness analysis of the previously described double layer models.

5 Discussion and Recommendations

The characteristics of the agent are described in three ways, first, the training characteristics are elaborated upon. Second, the NeuroBench results for the agents are presented. Lastly, the noise robustness is discussed. For the deployment of real closed-loop systems, these three aspects may be of varying importance. When training complex tasks, the training is a constraining factor. As discussed above, training SNN presents several additional challenges that may be detrimental. In these situations, ANN are still the algorithm of choice. Due to the challenging nature of training SNNs, sample efficiency and stability in training algorithms is a big requirement. Other reinforcement learning methods have shown to have improved efficiency and stability for training ANN. Examples of these are Proximal Policy Optimization (PPO)

and Soft Actor Critic (SAC) reinforcement learning. Next to these improved algorithms, future work should try to implement SNN specific methods such as reward-modulated STDP, which has been shown to reduce the effect of dead and saturated neurons during training [7], in the proven RL frameworks. Furthermore, eProp has also shown interesting results [10].

When analyzing the NeuroBench results, it can be observed that for resource constrained use cases, SNN may be a promising avenue. Their pruning opportunities allow for a significant reduction in complexity, even on hardware that is not specialized for neuromorphic algorithms. When training using RL, the SNN tends to struggle more with dead and saturated neurons, which allows for pruning without significant loss. Training ANN with similar architecture to the pruned ANN did not allow for the same performance. However, the pruning possibilities of the full ANN model were not explored.

To further analyze noise robustness of the models, more realistic noise characteristics should be injected. Currently, all inputs in the observation space receive the same white noise of mean 0 and a gain. In reality, the noise is usually smaller for sensors with a smaller range (such as the velocity). Furthermore, further work should analyze the behavior of leak and different neuron models under noise.

6 Conclusion

This article describes the opportunities and challenges of spiking neural networks (SNN) for closed-loop control, trained with actor-critic reinforcement learning. While the convergence of SNN during training was slow and noisy, the trained agents were able to control the CartPole. Evaluating the trained agents using NeuroBench highlights the use cases for ANN as well as SNN. Currently, the consistency and performance of ANN allow for a more reliable and performing controller in simulation. However, two aspects of the spiking agents offer interesting applications for systems deployed in real life. First, the computational resources required for SNN showed significantly lower than their non-spiking counterpart. These resources were further reduced by applying a simple pruning algorithm, leading to a 25x reduction in operations required per control action. This allows the agent to be run on extremely low power devices, or alternatively run the model with a much higher frequency if necessary. Next, the presence of noise in the observation of the agent severely affected the ANN models. At relatively low gains of the noise inserted, the SNN trained with temporal dynamics ($\beta > 0$) outperformed the ANN. Therefore, in real systems, the expected noise should be analyzed before the deciding on spiking or non-spiking controllers.

Further research is needed to improve the training efficiency of SNN using RL. Techniques such as reward-modulated STDP and eProp show promise in this regard. Additionally, the noise characteristics showed promising results for real-world applications. Future work could use the presented methods to deploy a controller in the real world and analyze their performance. Overall, the exploration of spiking agents in reinforcement learning is an interesting avenue for closed-loop control on deployed systems and warrants further investigation.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting Fundamentals of Experience Replay, July 2020. arXiv:2007.06700 [cs, stat].

[3] Ding Chen, Peixi Peng, Tiejun Huang, and Yonghong Tian. Deep Reinforcement Learning with Spiking Q-learning. *arXiv.org*, 2022. ARXIV_ID: 2201.09754 S2ID: 0a3104f2ca2308ac9930dd57cfbbe112d04f841d.

[4] Guisong Liu, Wenjie Deng, Xiurui Xie, Li Huang, and Huajin Tang. Human-Level Control Through Directly Trained Deep Spiking Q-Networks. *IEEE transactions on cybernetics*, pages 1–12, January 2022. ARXIV_ID: 2201.07211 MAG ID: 4294691690 S2ID: 2190a17a5e937c065adc5c139b563026ac174136.

[5] Luca Zanatta, Francesco Barchi, Andrea Bartolini, and Andrea Acquaviva. Artificial versus spiking neural networks for reinforcement learning in UAV obstacle avoidance. *ACM International Conference on Computing Frontiers*, May 2022. MAG ID: 4229040499 S2ID: 0903d078a954427d8c875da922d52d187981b958.

[6] Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. Spike-timing-dependent plasticity: A comprehensive overview. *Frontiers in Synaptic Neuroscience*, 4, 2012.

[7] Răzvan V. Florian. A reinforcement learning algorithm for spiking neural networks. *Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 299–306, September 2005. MAG ID: 2120905747 S2ID: cb6442b823c13339446a21fcb089428ec521a34c.

[8] Yuxiang Liu and Wei Pan. Spiking Neural-Networks-Based Data-Driven Control. *Electronics*, 12(2):310, January 2023. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.

[9] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions*

on Systems, Man, and Cybernetics, SMC-13(5):834–846, 1983.

[10] Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *bioRxiv*, 2019:738385, August 2019. MAG ID: 2967417697 S2ID: 858549b00245aadc92f91a2540f01398f5f389ae.

[11] Jianhao Ding, Zhaofei Yu, Yonghong Tian, and Tiejun Huang. Optimal ann-snn conversion for fast and accurate inference in deep spiking neural networks. *arXiv preprint arXiv:2105.11654*, 2021.

[12] Ding Chen, Peixi Peng, Tiejun Huang, and Yonghong Tian. Deep Reinforcement Learning with Spiking Q-learning. *arXiv.org*, 2022. ARXIV_ID: 2201.09754 S2ID: 0a3104f2ca2308ac9930dd57cfbbe112d04f841d.

[13] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

[14] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning, June 2016. arXiv:1602.01783 [cs] version: 2.

[15] Stein Stroobants, Julien Dupeyroux, and Guido C. H. E. de Croon. Neuromorphic computing for attitude estimation onboard quadrotors. *Neuromorphic Computing and Engineering*, 2(3):034005, September 2022. arXiv:2304.08802 [cs].

[16] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation, October 2018. arXiv:1506.02438 [cs].

A.2. Deploying SNN on the Bebop Parrot 2

Taking the method developed in section A.1, an SNN was trained to land the Parrot Bebop 2 drone. This work was presented at ICNCE 2024.

Control using Spiking Neural Networks Trained with Reinforcement Learning and Surrogate Gradients

K. Van den Berghe, S. Stroobants, G.C.H.E. de Croon

Delft University of Technology

Motivation

Deep reinforcement learning (RL) enables autonomous agents to achieve human-level performance in complex tasks through interaction with their environment. When deploying agents on edge devices, we are limited by computational and energy resources. Neuromorphic solutions such as spiking neural networks (SNNs) offer a promising avenue for improved energy efficiency.

We investigate **efficient deep spiking reinforcement learning**. Using **surrogate gradient** backpropagation through time, enabled by **training on sequences**, we take the networks' inherent memory into account.

Methods

- **Advantage Actor Critic (A2C):** an **on-policy** RL algorithm. Instead of batching the interactions of multiple actors to single transitions, the network trains on **full sequences**.
- **Spiking neural network:** continuous data is directly passed to the first hidden layer by using a linear encoding layer. The output of the network is defined by the membrane potentials of the final layer, which is a non-spiking LI neuron. The neurons in the hidden layers are **LIF neurons**.
- **Target tasks:** two tasks were analyzed, the **carpole task** and a **drone landing task**. In cartpole, the net controls left and right movement. In the drone landing task, the network applies a throttle command based on sonar altitude inputs.

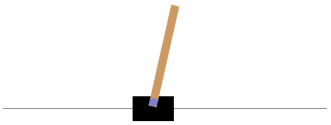


Figure 1. The Cartpole environment from Gymnasium.

Figure 2. The Parrot Bebop 2 drone is used for evaluation of the drone environment.

Training and Pruning Process

The SNN converges slower and noisier than comparable ANN. When the leakage can be learned, the network **learns to disregard the temporal dimension** (complete leakage). In a second experiment, the leak was fixed to a non-zero value.

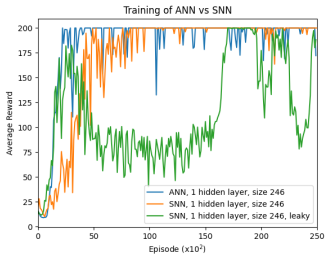


Figure 3. Training of an ANN and SNN with one hidden layer of size 246. The first SNN learns to reduce β to zero. The second SNN has a fixed leak, β = 0.65.

SNNs were **pruned based on activity**, removing dead and saturated neurons. This leads to reduced models.

- Hidden layer of **only 11 neurons** for the model with full leakage.
- Hidden layer of **only 21 neurons** for the model which has a fixed leakage.

These reduced models showed little decrease in performance.

NeuroBench Results

NeuroBench [1] is a community driven neuromorphic benchmarking framework. The complexity metrics provided by the benchmark include the synaptic operations, activation sparsity and footprint. The performance of the algorithms are described by the reward and the risk, representing the 5% worst performing interactions.

Baseline	ANN	SNN	SNN_{leaky}	SNN_{p}	$SNN_{leaky,p}$
Reward ($\mu \pm \sigma$)	1744 ± 1385	1620 ± 1600	232 ± 65	1570 ± 1480	200 ± 64
Risk	190	148	152	140	73
Footprint (bytes)	7.9×10^3	7.9×10^3	7.9×10^3	360	640
Activation Sparsity	0.0	0.68	0.92	0.49	0.65
SynOps Dense	1.7×10^3	1.7×10^3	1.7×10^3	66	126
SynOps Eff_MACs	1.5×10^3	0.9×10^3	0.9×10^3	44	84
SynOps Eff_ACs	0	238	59	12	16

Table 1. NeuroBench results for the single layer ANN and SNNs (regular, leaky, pruned (p), and leaky pruned (p) controlling the CartPole task.

The pruned spiking neural networks allow for an impressive **reduction in synaptic operations for a relatively small decrease in performance**. Pruning efforts for the ANN were unable to achieve similar characteristics.

Noise Robustness

The 5 different models, with artificial neurons, spiking neurons with and without temporal capabilities, are compared upon injection of Gaussian noise into the input data.

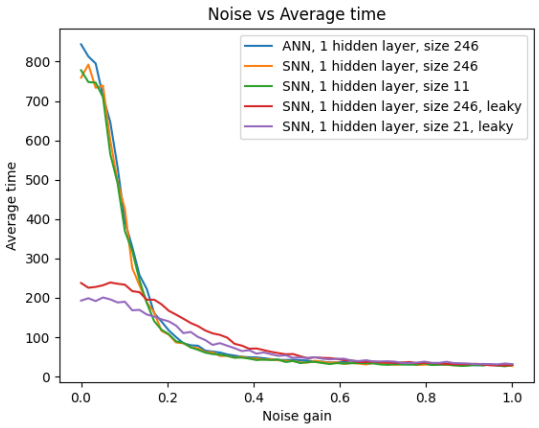


Figure 4. Noise robustness analysis of the artificial, and spiking models.

- **Pruned models achieved similar performance** to their original models independent of noise injection.
- **Leakage enables the models to perform more robustly** to noisy sensory data.

Deployment on a UAV

A model with two hidden layers of 32 LIF neurons, uses noisy sonar altitude measurements, requiring the network to estimate velocity. The net controls the throttle setting, to successfully land a UAV.

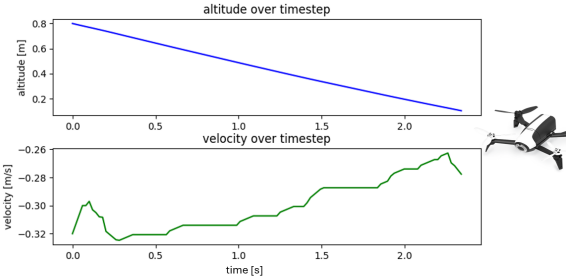


Figure 5. An SNN trained with the previously described pipeline can successfully land the Parrot Bebop 2 drone, decreasing velocity as it nears the ground.

Future Work

While it is possible to train an SNN by modifying the A2C algorithm to train on sequences and using the surrogate gradient for backpropagation, the SNN fails to fully exploit temporal dynamics. In future work, I would like to explore how to **train an SNN to exploit temporal dynamics with RL** and attempt to **combine the fast learning of ANN as a critic or guiding policy with a SNN-based actor** in RL.

References

[1] Jason Yik and Korneel Van den Berghe et al.