SÉRGIO FERNANDOVITCH CHEVTCHENKO

Reinforcement Learning with Spiking Neural Networks

Recife
2023

SÉRGIO FERNANDOVITCH CHEVTCHENKO

Reinforcement Learning with Spiking Neural Networks

Tese apresentada ao Programa de Pós-Graduação em Ciências da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciências da Computação.

Área de Concentração: Inteligência Computacional.

Orientadora: Dr. Teresa Bernarda Ludermir

Recife
2023

Catalogação na fonte
Bibliotecária Mônica Uchôa, CRB4-1010

C529r Chevtchenko, Sérgio Fernandovitch
Reinforcement learning with spiking neural networks / Sérgio Fernandovitch Chevtchenko. – 2023.
117 f.: il., fig., tab.

Orientadora: Teresa Bernarda Ludermir.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2023.
Inclui referências e apêndices.

1. Aprendizagem por reforço. 2. STDP. 3. Redes neurais de impulsos. 4. FEAST. 5. ODESA. I. Ludermir, Teresa Bernarda. (Orientadora). II. Título.

004 CDD (23. ed.) UFPE - CCEN 2023 – 208

Sergio Fernandovitch Chevtchenko

“Reinforcement Learning with Spiking Neural Networks”

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação. Área de Concentração: Inteligência Computacional.

Aprovada em: 15/08/2023.

_____

Orientadora: Profa. Dra. Teresa Bernarda Ludermir

BANCA EXAMINADORA

_____

Prof. Dr. Cleber Zanchettin
Centro de Informática/ UFPE

_____

Prof. Dr. Alexandre Marcireau
Centre for Neuromorphic Systems / Western Sydney University

_____

Prof. Dr. Saeed Afshar
Centre for Neuromorphic Systems / Western Sydney University

_____

Profa. Dra. Anna Helena Reali Costa
Departamento de Engenharia de Computação e Sistemas Digitais / USP

_____

Prof, Dr. Denis Deratani Mauá
Departamento de Ciência da Computação / IME/ USP

ACKNOWLEDGEMENTS

Special thanks go to my advisor, Teresa Bernarda Ludermir for her patience, motivation and dedication. Warm appreciation is extended to Saeed Afshar and Yeshwanth Bethi, whose insightful discussions and valuable contributions have significantly enriched this research. I would like to show my sincere gratitude to my teachers, both in schools and universities. Last but by no means least, thanks to my family and friends for support and encouragement!

ABSTRACT

Artificial intelligence systems have made impressive progress in recent years, but they still lag behind simple biological brains in terms of control capabilities and power consumption. Spiking neural networks (SNNs) seek to emulate the energy efficiency, learning speed, and temporal processing of biological brains. However, in the context of reinforcement learning (RL), SNNs still fall short of traditional neural networks. The primary aim of this work is to bridge the performance gap between spiking models and powerful deep RL (DRL) algorithms on specific tasks. To this end, we have proposed new architectures that have been compared, both in terms of learning speed and final accuracy, to DRL algorithms and classical tabular RL approaches. This thesis consists of three stages. The initial stage presents a simple spiking model that addresses the scalability limitations of related models in terms of the state space. The model is evaluated on two classical RL problems: grid-world and acrobot. The results suggest that the proposed spiking model is comparable to both tabular and DRL algorithms, while maintaining an advantage in terms of complexity over the DRL algorithm. In the second stage, we further explore the proposed model by combining it with a binary feature extraction network. A binary convolutional neural network (CNN) is pre-trained on a set of naturalistic RGB images and a separate set of images is used as observations on a modified grid-world task. We present improvements in architecture and dynamics to address this more challenging task with image observations. As before, the model is experimentally compared to state-of-the-art DRL algorithms. Additionally, we provide supplementary experiments to present a more detailed view of the connectivity and plasticity between different layers of the network. The third stage of this thesis presents a novel neuromorphic architecture for solving RL problems with real-valued observations. The proposed model incorporates feature extraction layers, with the addition of temporal difference (TD)-error modulation and eligibility traces, building upon prior work. An ablation study confirms the significant impact of these components on the proposed model's performance. Our model consistently outperforms the tabular approach and successfully discovers stable control policies in *mountain car*, *cart-pole* and *acrobot* environments. Although the proposed model does not outperform PPO in terms of optimal performance, it offers an appealing trade-off in terms of computational and hardware implementation requirements: the model does not require an external memory buffer nor global error gradient computation, and synaptic updates occur online, driven by local learning rules and a broadcast TD-error signal. We conclude by highlighting the limitations of our approach and suggest promising directions for future research.

Keywords: reinforcement learning; STDP, spiking neural networks; FEAST; ODESA.

RESUMO

Nos últimos anos, sistemas de inteligência artificial têm progredido de forma impressionante, mas ainda estão aquém de cérebros biológicos simples em termos de capacidades de controle e consumo de energia. As redes neurais de impulsos (SNNs) buscam emular a eficiência energética, velocidade de aprendizado e processamento temporal de cérebros biológicos. No entanto, no contexto de aprendizado por reforço (RL), as SNNs ainda ficam aquém das redes neurais tradicionais. O objetivo principal deste trabalho é aproximar em termos de desempenho os modelos SNN dos algoritmos de aprendizagem profunda por reforço (DRL) em tarefas específicas. Para isso, propomos novas arquiteturas que foram comparadas, tanto em termos de velocidade de aprendizado quanto de precisão final, com algoritmos DRL e abordagens RL tabulares clássicas. Esta tese consiste em três etapas. A etapa inicial apresenta um modelo simples de uma rede de impulsos que aborda as limitações de escalabilidade de modelos relacionados em termos do espaço de estados. O modelo é avaliado em dois problemas clássicos de RL: *grid-world* e *acrobot*. Os resultados sugerem que o modelo proposto é comparável ao algoritmo tabular e a DRL, mantendo uma vantagem em termos de complexidade sobre o algoritmo DRL. Na segunda etapa, exploramos mais o modelo proposto, combinando-o com uma rede binária para extração de características. Uma rede neural convolucional (CNN) binária é pré-treinada em um conjunto de imagens RGB naturalistas e um conjunto separado de imagens é usado como observações em uma ambiente modificado de *grid-world*. Melhorias na arquitetura e na dinâmica são apresentadas para tratar esse problema mais complexo, com observações de imagens. Como antes, o modelo é comparado experimentalmente com algoritmos DRL do estado da arte. Além disso, experimentos complementares são fornecidos com objetivo de apresentar uma visão mais detalhada da conectividade e plasticidade entre diferentes camadas da rede. A terceira etapa desta tese apresenta uma nova arquitetura neuromórfica para resolver problemas de RL com observações de valores reais. O modelo proposto incorpora camadas de redução de dimensionalidade, com a adição de modulação por *TD-error* e *eligibility traces*, baseando-se em trabalhos anteriores. Um estudo adicional é focado em confirmar o impacto significativo desses componentes no desempenho do modelo proposto. O modelo supera consistentemente a abordagem tabular e descobre com sucesso políticas de controle estáveis nos ambientes *mountain car*, *cart-pole* e *acrobot*. Embora o modelo proposto não supere o PPO em termos de latência, ele oferece uma alternativa em termos de requisitos computacionais e de hardware: o modelo não requer um buffer de memória externo nem computação de gradiente de erro global. Além disso, as atualizações sinápticas ocorrem online, por meio de regras de aprendizado local e um sinal de erro global. A tese conclui apresentando limitações da pesquisa e sugestões de trabalhos futuros.

Palavras-chave: aprendizagem por reforço; STDP, redes neurais de impulsos; FEAST; ODESA.

LIST OF FIGURES

LIST OF SYMBOLS

A_+	Scaling factor for the positive eligibility trace
A_-	Scaling factor for the negative eligibility trace
α	Step-size parameter for dutch traces
a_+	Amplitude of excitatory connectivity
a_-	Amplitude of inhibitory connectivity
C_m	Membrane capacitance
δ	Temporal difference (TD) error
$\Delta\theta_i$	The change in threshold for neuron i
$\Delta\overrightarrow{w_i}$	The change in weights for neuron i
$e(s,t)$	Eligibility trace for state s at time t
ϵ	Exploration probability
η	Learning rate for weights
η_a	The learning rate for the actor
η_c	The learning rate for the critic
η_{td}	The learning rate for the temporal difference (TD) error updates
η_{th}	Learning rate for the threshold
$f_i(t)$	Firing function for neuron i
γ	Discount factor, ranging from 0 to 1
$I(t)$	Input current at time t
λ	Decay factor for the eligibility trace
N_h	Number of hidden neurons
N_i	Number of input neurons
n_+	Number of excitatory synapses, expressed as a percentage of N_h
n_-	Number of inhibitory synapses, expressed as a percentage of N_h

$P^+ij(t)$	Positive eligibility trace between neurons i and j
$P^-ij(t)$	Negative eligibility trace between neurons i and j
π	Policy, a function that specifies what action the agent should take at each state.
$Q(s_t, a_t)$	Q-table value for a given state-action pair (s_t, a_t)
$\overrightarrow{c_a}i$	Synaptic eligibility traces for the actor neuron i
$\overrightarrow{c_c}$	Synaptic eligibility traces for the critic
$r(t)$	Reward signal at time t
R_m	Membrane resistance, reflecting the diffusion of ions through the membrane
s	State in the environment.
τ_+	Time constant to modulate the decaying time of the positive eligibility trace
τ_-	Time constant to modulate the decaying time of the negative eligibility trace
τ_e	Time constant for the decay of the eligibility trace
τ_m	Membrane discharge time constant
τ_s	Synaptic tag discharge time constant
θ_i	Threshold value for the neuron i
θ_o	Parameter used to increase the threshold of the neuron
V_{reset}	Reset voltage after spike emission
V_{th}	Voltage threshold for spike emission
$V^\pi(s)$	Value function for a policy π at state s.
V_m	Voltage potential of a LIF neuron
$v_i(t)$	Potential of neuron i at time t
$w_{j,i}$	Synaptic efficacy between neurons i and j
$\overrightarrow{x}$	Input signal

Y_t^Q	Target value for the Q-function at time step t.
$Z_{j,i}$	Eligibility trace of the synapse between neurons j and i
ζ_{ij}	Instantaneous eligibility trace change factor between neurons i and j
$\xi_i(t)$	Gaussian noise applied to neuron i at time t
$\mathbb{E}_\pi$	Expected value following policy π

.

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Many of the recent advances in machine learning are inspired by nature. The present work aims to bring closer two fields rooted in insights from neuroscience and biology: reinforcement learning (RL) and spiking neural networks (SNNs). Reinforcement learning is a paradigm that aims to reproduce the way animals learn (SUTTON; BARTO, 2018). In contrast to a supervised learning scenario, an RL agent can be trained to perform a task without an expert teacher or clear, step-by-step knowledge of how to accomplish the goal. It is generally enough for the agent to be able to repeat trials and to receive a reward after a successfully executed series of actions. Note the similarity to, for instance, how a laboratory mouse can be trained to navigate a maze with food as a reward. While the foundations of RL were laid three decades ago (SUTTON, 1988), the advancements in deep learning (DL) in the past decade have enabled training RL agents for increasingly more complex tasks. Some recent examples of deep reinforcement learning (DRL) applications include playing Atari games (MNIH et al., 2015), learning agile and dynamic motor skills for legged robots (HWANGBO et al., 2019) and training robotic manipulators from scratch (GU et al., 2017; ANDRYCHOWICZ et al., 2020). However, engineered systems are still far behind their animal counterparts in the ability to quickly learn a new and complex task. While DRL enables mastery of a specific skill on a high computational budget, nature has evolved organisms capable of rapid adaptation and learning new skills (CULLY et al., 2015).

This limitation is partially due to the training process of artificial neural networks (ANNs), which constitute the brain of DRL agents. Training such networks usually involves a large collection of examples stored in a separate memory. Furthermore, an already trained network would also typically require significantly more power than biological brains, constraining applications on devices with embedded electronics or an otherwise limited power supply (BING et al., 2018b).

Traditional ANNs use continuous signals to approximate an unknown function. In contrast, SNNs use discrete spikes and process information over time. The use of single-wire synapses allows the implementation of SNNs in low-power neuromorphic chips, capable of simulating up to 10^8 spiking neurons in real time (THAKUR et al., 2018; FRADY et al., 2020). Such neuromorphic hardware is also capable of low-latency sensory response when using appropriate sensors, such as a dynamic vision sensor (DVS) (GALLEGO et al., 2019). An implementation based on optical hardware has been shown to further reduce latency and improve the simulation speed of smaller spiking networks (FELDMANN et al., 2019).

SNNs can be trained using a biologically inspired algorithm known as spike-timing-dependent plasticity (STDP). An advantage of this method is that each synapse is only required to be aware of the spikes produced by presynaptic and postsynaptic neurons. By

combining STDP with a global reinforcement signal (R-STDP), a spiking agent can solve RL tasks without computing a global error gradient.

1.1 RESEARCH PROBLEM

While there have been recent advancements in spiking networks for reinforcement learning, SNNs have not been previously shown to be on par with the state-of-the-art when solving RL tasks in terms of speed and quality of learning. One of the challenges is a trade-off between the biological plausibility and the complexity of the model. Spiking models designed to mimic a brain's functionality tend to have a higher computational cost of simulation and, thus, not be applicable to practical problems. Even spiking networks with simple neural models, such as Leaky Integrate-and-Fire (LIF), require additional complexities to achieve a balanced and stable activity. Thus, in the present work, we choose a compact model of a spiking network, aimed at a low computational cost of simulation and eventual implementation on a hardware platform.

1.2 RESEARCH GOALS

The main objective of the present work is to propose a spiking neural network intended to solve specific RL tasks comparably to other commonly used RL algorithms, both in terms of speed and control efficiency.

1.2.1 Specific goals

- Propose a spiking model aimed at hardware implementation. To this end, we use local synaptic plasticity rules that do not require global computations while also avoiding rate-based neural models, as these can be viewed as approximations of analog neurons and require additional complexity for implementation.
- Demonstrate competitive performance on classical RL problems when compared to state-of-the-art non-spiking algorithms.
- Expand the range of possible applications by proposing solutions to improve state-space scalability and learning from continuous rewards.

1.3 RESEARCH CONTRIBUTIONS

The work presented in this thesis contributes to the growing body of research on reinforcement learning and spiking neural networks. A set of novel neuromorphic architectures are proposed to solve increasingly complex RL tasks. This architectures are aimed at low memory and hardware requirements, and are compared to state-of-the-art DRL models at benchmark tasks.

The following works have been published or submitted for publication within the scope of this graduate program:

- "Learning from Sparse and Delayed Rewards with a Multilayer Spiking Neural Network" — International Joint Conference on Neural Networks (IJCNN), 2020 (CHEVTCHENKO; LUDERMIR, 2020).
- "Combining STDP and binary networks for reinforcement learning from images and sparse rewards" — Neural Networks 144, 496–506, 2021 (CHEVTCHENKO; LUDERMIR, 2021).
- "Combining PPO and Incremental Conductance for MPPT under Dynamic Shading and Temperature" — Applied Soft Computing 131, 109748, 2022 (CHEVTCHENKO et al., 2022).
- "A Neuromorphic Architecture for Reinforcement Learning from Real-Valued Observations" — preprint available on arXiv (CHEVTCHENKO et al., 2023).

1.4 THESIS OUTLINE

Chapter 2 provides a background overview of RL, the spiking neural model and reward-based synaptic plasticity. Chapter 3 describes the first iteration of the proposed SNN. This model uses a simplified version of reward-modulated STDP to solve two classical RL tasks and address a scalability issue identified in related spiking models. The proposed network is compared to a classical and a deep reinforcement learning algorithm in terms of speed and latency. Chapter 4 evaluates a modified version of the previously proposed network in combination with a binary feature extractor. The proposed architecture is demonstrated to be a competitive alternative to deep reinforcement learning (DRL) in the evaluated environment with image observations, and it provides a foundation for more complex future applications of spiking networks. Chapter 5 proposes a neuromorphic architecture for addressing RL tasks with real-values inputs. Finally, Chapter 6 concludes the thesis by summarizing the main findings and contributions to the field. Additionally, this chapter highlights the limitations of the present work and delineates possible directions for future research.

2 BACKGROUND

2.1 REINFORCEMENT LEARNING

The concepts of reinforcement learning presented in this section are based mainly on the RL book by Sutton e Barto (2018), which should be consulted for further details.

2.1.1 Problem Formulation

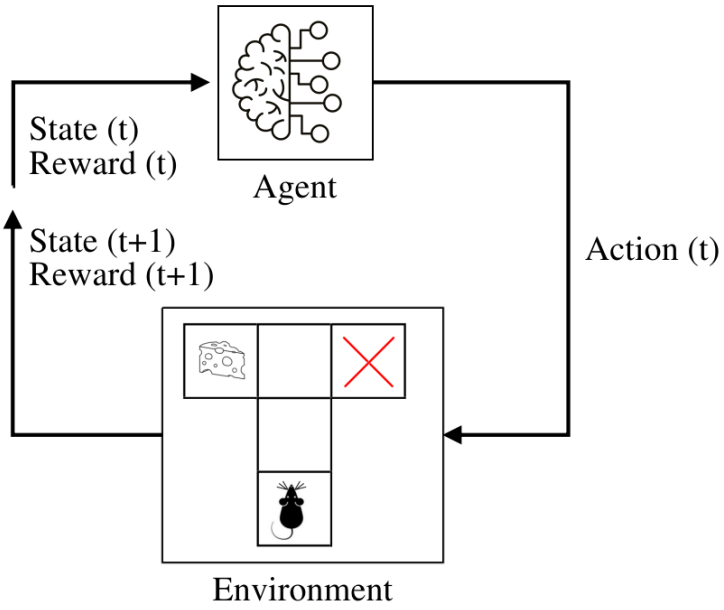
A reinforcement learning (RL) problem can be described as the interaction between an agent and its environment. At each discrete timestep t, the agent needs to decide on an action $a(t)$ to take based on its current observation of the environment, represented by a state $s(t)$, and the associated reward $r(t)$. The environment processes the action $a(t)$, resulting in a new state $s(t+1)$ and a reward $r(t+1)$. This process continues over time, generating a sequence of states, actions, and rewards. This setup is illustrated in Figure 1.

A sequence of actions and state observations is called an episode. Considering and episode with duration of t steps, the goal of an agent is to find a sequence of actions $a(1), a(2), \ldots, a(t)$ that maximizes the expected total rewards during an episode. The agent also has to balance the trade-off between exploration and exploitation. Exploration usually refers to the agent trying more random actions in order to gather information about the environment. Exploration can also be conducted in a structured manner, where certain areas of the state-action space are targeted. Exploitation refers to choosing actions that are known to have produced good results in the past. Efficient learning requires balancing these two competing demands. Typically, an agent starts taking random actions and, as the episode progresses, the probability of taking a random action is decreased to a minimum.

2.1.2 Markov decision process

Most RL problems are formulated within the framework of a Markov Decision Process (MDP). An environment is considered to be Markovian if the current state observation $s(t)$ and action $a(t)$ contain all the information necessary to compute the transition to the next state $s(t+1)$, and predict the associated reward $r(t+1)$. In other words, an agent in a Markovian environment does not need to remember previous states before $s(t)$ in order to determine the appropriate action $a(t)$. As an example, consider the environment illustrated in Figure 1. In this environment, the agent only needs to know its current position to unambiguously decide on the next action. This is because the dynamics of the environment depend solely on the current state and action, and not on the history of past states and actions. It is worth noting that not all real-world environments are

Figure 1 – An illustration of the interaction between an agent and the environment in a RL problem



Source: The author, 2023

Markovian, and additional techniques may be necessary to handle non-Markovian aspects of an environment.

An MDP is defined by tuple (S, A, P, R), where S is a set of states of the environment. In the context of the present thesis, S is assumed to be finite. In an MDP, a state $s \in S$ contains all the relevant information required to make decisions at any given time step t. A state transition function $P(s(t+1)|s(t), a(t))$ defines the probability of transitioning from state $s(t)$ to state $s(t+1)$ when the agent takes action a. A reward function $R(s(t), a(t), s(t+1))$ defines the immediate reward $r(t+1)$ the agent receives after taking the action $a(t)$ in state $s(t)$ and transitioning to state $s(t+1)$. A is a set of actions the agent can take in response to the current state. As with the set of states S, A is assumed to be finited in the present work. Additionally, a policy π defines a transition from states to actions, specifying which action to take in each state. An optimal policy π^* is one that achieves the highest possible expected return from any starting state. In other words, by following the optimal policy the agent obtains the maximum cumulative reward over time.

Considering an agent that starts in state s and follows a specific policy π for future steps, a value function $V(s)$ is used to represent the expected cumulative discounted reward of a state s. The value function $V^\pi(s)$ for a policy π is defined as:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r(t+k+1) \middle| s(t) = s \right], \tag{2.1}$$

where $\mathbb{E}_\pi$ denotes the expected value from following a policy π and γ is the discount

factor ($0 \leq \gamma \leq 1$) that determines the relative importance of future rewards compared to immediate rewards. If γ is set to 0, the agent is considered "myopic" and aims only at maximizing the immediate reward $r(t+1)$. On the other hand, taking γ closer to 1 gives importance to future rewards.

When the agent employs policy π to interact with the environment while simultaneously estimating its value using Equation 2.1, this is considered as an on-policy method (SUTTON; BARTO, 2018). On the other hand, off-policy methods estimate the value of a policy different from the one being used to generate the data. A common off-policy method is Q-learning (WATKINS, 1989), which estimates the optimal action-value function, regardless of the policy the agent is following. This algorithm is used as a baseline in the Chapter 3 and is described in more detail in Section 3.4.2.1.

Formally, an off-policy method uses a behavior policy μ to generate behavior, while it evaluates or improves a target policy π. The key advantage of off-policy methods is that they can learn from a wide variety of data, including historical or exploratory actions. However, this can result in slower convergence, as the historical data can present a higher variance (SUTTON; BARTO, 2018).

2.1.3 Temporal Difference Learning

Temporal Difference (TD) learning is a RL method that uses ideas from dynamic programming to learn an optimal policy in an online manner. The key idea behind TD learning is to update the value estimates based on the difference between the current value estimate and a new, more informed estimate obtained after observing a new state and reward. This difference is called the temporal difference error δ:

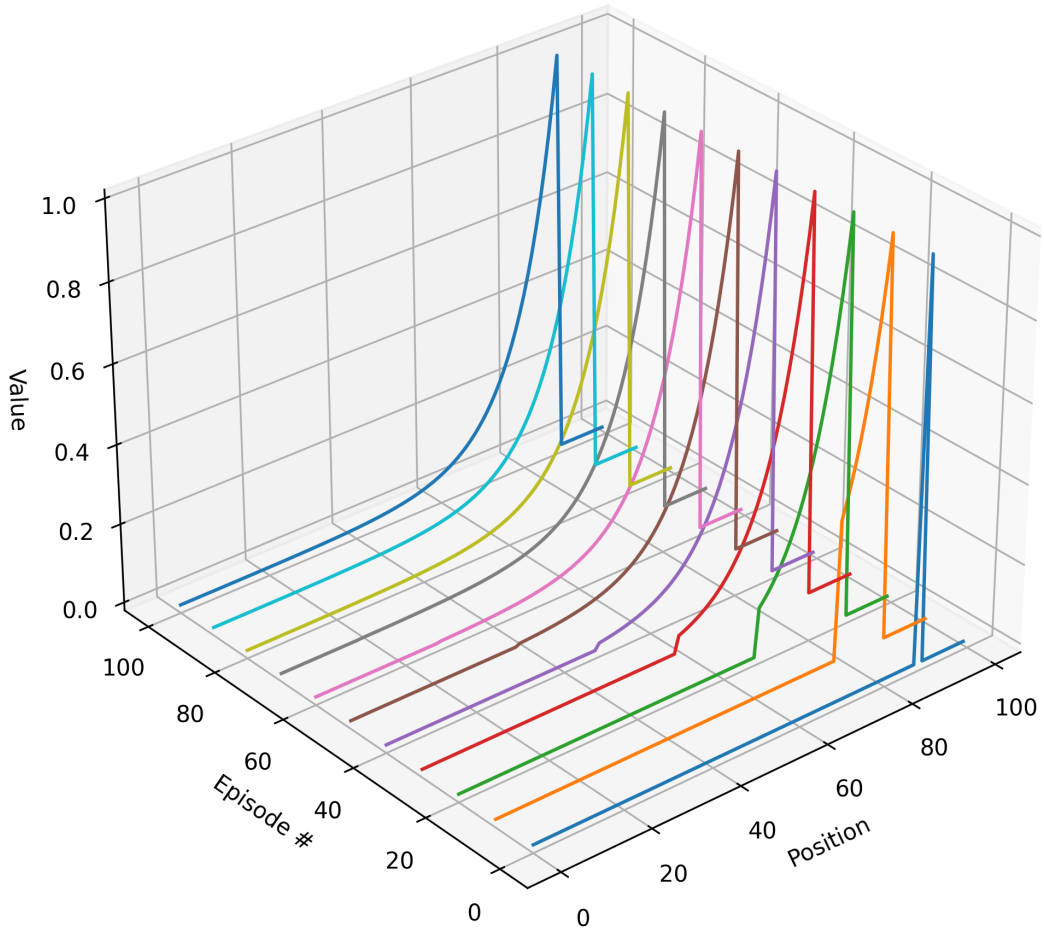
$$\delta = r(t+1) + \gamma V(s(t+1)) - V(s(t)), \tag{2.2}$$

where $V(s(t))$ and $V(s(t+1))$ are the value estimates for the current state and the next state, respectively, as defined by Equation 2.1. The value function estimates are then updated using the TD error and a learning rate η:

$$V(s(t)) \leftarrow V(s(t)) + \eta\delta. \tag{2.3}$$

In order to illustrate the above rule, consider a linear track environment. The agent starts at position 0 and travels one unit to the right at each timestep. There is a single reward of +1 at position 90 and the episode ends at position 100, when the agent is returned to the initial coordinate. Assuming that V is initialized by zeroes, during the first episode the TD error from Equation 2.2 is 0, except at position 90. Thus, at the first episode, the value function is updated at timestep preceding the reward. The expected reward then propagates backwards during next episodes. This process is depicted in Figure 2, by using learning rate $\eta = 1$ and the temporal difference error $\delta = 0.9$.

Figure 2 – An illustration of a value function update using TD-learning without eligibility traces



Source: The author, 2023

2.1.4 Eligibility Traces

The TD learning model described by Equations 2.2 and 2.3 performs an update to the last visited state $s(t)$, after calculating the error δ at state $s(t+1)$. In case of an environment with a single rewarded state, such as the linear track from the previous example, this means that a value update will be performed sparsely, leading to longer learning times. This phenomenon is primarily due to the "temporal credit assignment problem" in TD learning. Note that the value of a single state is updated after the first episode in Figure 2. The sparseness of the reward updates in such scenarios prevents the algorithm from efficiently propagating the reward information back through the sequence of states.

A potential solution to overcome this limitation is through the use of eligibility traces, a biologically inspired mechanism that allows the algorithm to update the value estimates of not only the most recent state but all the previously visited states, proportionally to their temporal proximity to the reward. This results in a faster propagation of the reward

signal and value update. The value of a state $s(t)$ is updated by a weighted TD error $\delta(t)$ with the eligibility trace $e(t)$:

$$V(s(t)) \leftarrow V(s(t)) + \eta e(s,t)\delta, \tag{2.4}$$

where $e(s,t)$ is the eligibility trace for state $s(t)$, η is the learning rate, and δ is the TD error as per Equation 2.2. Eligibility traces decays with a factor λ between 0 and 1:

$$e(s,t) = \lambda e(s,t-1). \tag{2.5}$$

Note that eligibility traces can be updated using various methods, the most common being the "replacing traces", "dutch traces" and "accumulating traces". In the accumulating traces method, the eligibility trace for a state increases every time the state is visited. However, in problems where states can be visited frequently within the same episode, the eligibility trace can become excessively large, which could lead to unstable learning.

The replacing traces method addresses this problem by setting the eligibility trace for a visited state to 1, instead of incrementing it. This method can be more suitable for tasks where states are frequently revisited and is used throughout the present work. Figure 3 illustrates the value update in the linear track environment, with added replacing traces. Note that the value update is significantly faster and smoother than by just back-propagating the temporal difference.

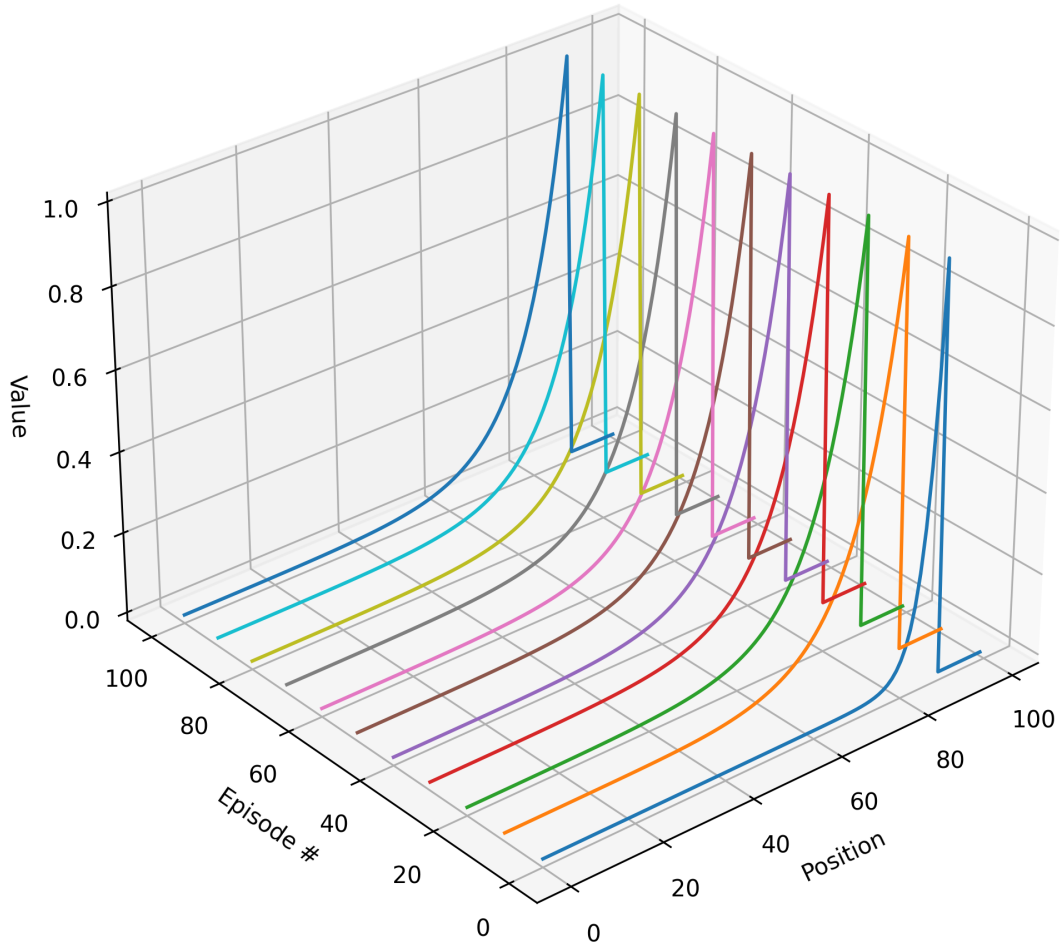
The dutch traces update finds a middle ground by using a step-size parameter α:

$$e(s,t) = (1-\alpha)\lambda e(s,t-1) + 1. \tag{2.6}$$

If the parameter α is set to zero, the dutch trace becomes equivalent to accumulating trace. Conversely, setting α to 1 makes the equation above into replacing traces.

Besides being used in reinforcement learning, eligibility traces are extensively used in spiking neural networks. In this context, eligibility traces act like as a distributed memory of recent neural activity. The present work explores spiking models with eligibility traces for feature extraction and temporal credit assignment.

Figure 3 – A value function update on a linear track using TD-learning with eligibility traces



Source: The author, 2023

2.2 SPIKING NEURAL NETWORKS

Spiking Neural Networks (SNNs) are a class of artificial neural networks that aim at emulating aspects of the biological processes observed in the brain more closely than traditional models. SNNs draw their inspiration from the fact that biological neurons do not continuously transmit information, but instead, generate discrete spikes of activity. This action is leveraged in SNNs where information is communicated through sequences of spikes over time, rather than by constant real numbers as in standard neural networks. Moreover, SNNs combine memory and processing in a way that reflects the interconnected structure of the brain. By integrating both functions, SNNs can process and interpret temporal information more effectively, emulating the dynamic, time-dependent aspects of biological cognition. The following sections describe the most common spiking neural models, synaptic plasticity and how these can be used to solve distal reward problems.

2.2.1 Neural Models

2.2.1.1 Integrate and Fire

The integrate and fire (IF) is a simplified and computationally efficient model of a biological neuron. The voltage potential V_m of the neuron depends on the membrane capacitance C_m and the input current $I(t)$:

$$C_m \frac{dV_m(t)}{dt} = I(t). \tag{2.7}$$

For a discrete representation using Euler method for numerical integration with a time step Δt, the voltage potential update is given by:

$$V_m(t + \Delta t) = V_m(t) + \Delta t \left(\frac{I(t)}{C_m} \right). \tag{2.8}$$

A spike is emitted when the voltage reaches a certain threshold value V_{th}, in which case the potential resets to V_{reset}:

$$\text{if } V_m \geq V_{th}, \text{ then } \begin{cases} V_m \leftarrow V_{reset} \\ \text{emit spike} \end{cases} \tag{2.9}$$

2.2.1.2 Leaky Integrate and Fire

The leaky IF model introduces a "leak" term that reflects the diffusion of ions trough the membrane (ABBOTT, 1999):

$$C_m \frac{dV_m}{dt} = I(t) - \frac{V_m(t)}{R_m}. \tag{2.10}$$

2.2.1.3 Izhikevich

The Izhikevich neuron model (IZHIKEVICH, 2003) is able to reproduce various firing patterns observed in the nervous system, while retaining low computational complexity. The model is described by:

$$\begin{aligned} \frac{dv}{dt} &= 0.04v^2 + 5v + 140 - u + I, \\ \frac{du}{dt} &= a(bv - u), \end{aligned} \tag{2.11}$$

$$\text{if } v \geq 30\text{mV}, \text{ then } \begin{cases} v \leftarrow c, \\ u \leftarrow u + d, \end{cases} \tag{2.12}$$

where v is the potential of the neuron, u is the recovery variable, and a, b, c, d are constants than can be tuned to reproduce different spiking patterns.

2.2.2 Spike-timing-dependent Plasticity

Training of traditional ANNs usually involves gradient descent methods. Even when used in an RL framework, an ANN is trained through back-propagation on a sequence of previously observed samples from the environment (MNIH et al., 2015). While it is possible to train SNNs using modified gradient-based optimization, this does not leverage the low-power requirements of biological neurons (TAVANAEI et al., 2018). Instead of computing a global gradient over all of the synapses, SNNs can employ a biologically plausible process called spike-timing-dependent plasticity (STDP). Considering a connection w_{ij} between presynaptic neuron j and postsynaptic neuron i, this update is described by Equation 2.13:

$$\Delta w_{ij} = \begin{cases} A_+ e^{(\Delta t/\tau_+)} & \text{if } \Delta t > 0 \text{ (pre-before-post)} \\ -A_- e^{-\Delta t/\tau_-} & \text{if } \Delta t < 0 \text{ (post-before-pre)} \\ 0 & \text{otherwise,} \end{cases} \tag{2.13}$$

where Δw_{ij} is the change in synaptic weight, $\Delta t = t_{\text{post}} - t_{\text{pre}}$. A_+ and A_- are positive constants that define the maximum weight change for potentiation and depression, respectively. Time constants τ_+ and τ_- are used for potentiation and depression, respectively.

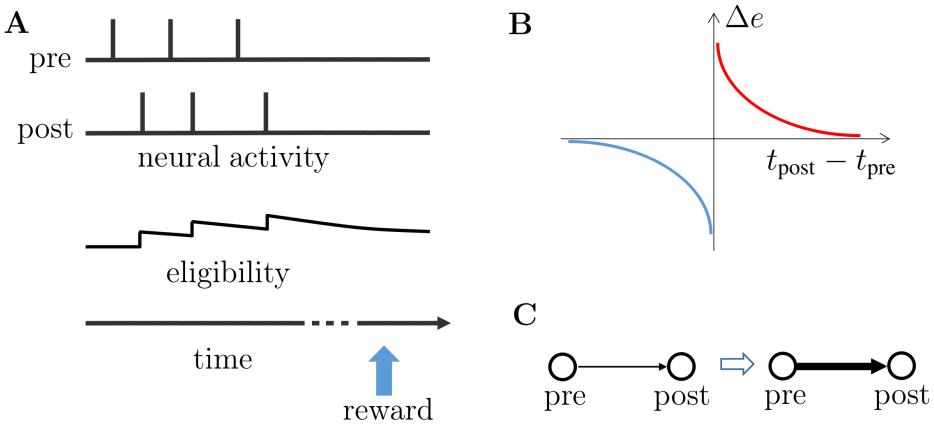
This synaptic plasticity rule only requires each synapse to be aware of corresponding pre- and post-synaptic neurons. STDP is illustrated in Figure 4 and can be applied to a network of neurons. In Figure 4 **(A)** presynaptic neuron repeatedly causes the postsynaptic neuron to fire, with a small delay between two events. The pre-post activation causes the eligibility flag to increase as a function of the delay between spikes, as illustrate in Figure 4 **(B)**, where the vertical axis indicates the strength of the eligibility change and the horizontal axis relates to the time between pre and post synaptic spikes. Finally, if a reward signal is provided after some time, the synaptic strength between pre and post neurons is increased proportionally to the accumulated eligibility trace (Figure 4 **(C)**). Additionally, STDP can be modulated by a global reward signal (R-STDP) and has been shown to solve reinforcement learning problems without the need for explicit gradient computation over the synapses. In the present work we focus on RL tasks with delayed and sparse rewards, resembling how animals are rewarded with food after a successful task completion.

2.2.3 Addressing RL Problems by Using SNNs and STDP

This final introductory section provides an overview of early works that have laid ground to the current research in using spiking architectures to solve reinforcement learning problems.

A rule for synaptic modulation by eligibility traces called MSTDPET is proposed by Florian (2007). Considering a discrete timestep of δt and a connection w_{ij} between

Figure 4 – An illustration of the reward-modulated spike-timing-dependent plasticity (R-STDP). **(A)** presynaptic neuron repeatedly causes the postsynaptic neuron to fire, with a small delay between two events. The pre-post activation causes the eligibility flag to increase as a function of the delay between spikes **(B)**. **(C)** If a reward signal is provided after some time, the synaptic strength between pre and post neurons is increased proportionally to the accumulated eligibility trace.



Source: The author, 2023

presynaptic neuron j and postsynaptic neuron i, the synaptic weight is updated as follows:

$$w_{ij}(t+\delta t) = w_{ij}(t) + \gamma r(t+\delta t) Z_{ij}(t+\delta t) \tag{2.14}$$

$$Z_{ij}(t+\delta t) = Z_{ij}(t) - \frac{Z_{ij}(t)}{\tau_e} + \zeta_{ij}(t) \tag{2.15}$$

$$\zeta_{ij}(t) = P_{ij}^{+}(t) f_i(t) + P_{ij}^{-}(t) f_j(t) \tag{2.16}$$

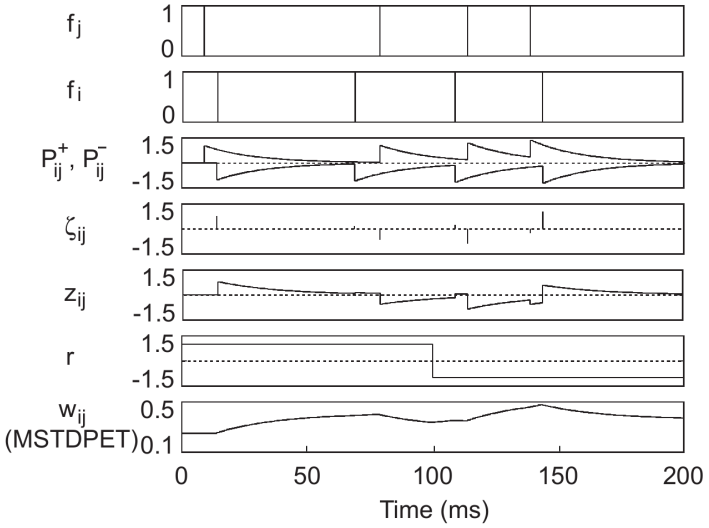
$$P_{ij}^{+}(t) = P_{ij}^{+}(t-\delta t) e^{-\delta t/\tau_+} + A_+ f_j(t) \tag{2.17}$$

$$P_{ij}^{-}(t) = P_{ij}^{-}(t-\delta t) e^{-\delta t/\tau_-} + A_- f_i(t), \tag{2.18}$$

where $f_i(t)$ is 1 when neuron i fires at time t and 0 otherwise. $Z_{ij}(t)$ is the eligibility trace, with decay time constant τ_e and instantaneous change factor $\zeta_{ij}(t)$. Time constants τ_+ and τ_- are used to modulate the decaying time of eligibility traces P_{ij}^{+} and P_{ij}^{-}. The MSTDPET update ensures that if neuron i consistently fires after the neuron j and a reward signal is received within a time window defined by τ_+, the connection between the two neurons is strengthened over time. On the other hand, a negative reward would act to decrease the probability of postsynaptic neuron i firing as a consequence of neuron j. This rule is illustrated in Figure 5, by using learning rate γ=0.2, τ_-=τ_+=20ms, A_+=1 and A_-=-1.

A plasticity rule similar to MSTDPET is independently proposed by Izhikevich (2007). It is experimentally demonstrated to reproduce Pavlovian conditioning in a randomly connected network of 1000 Izhikevich neurons. In this experiment, a 100 random sets of

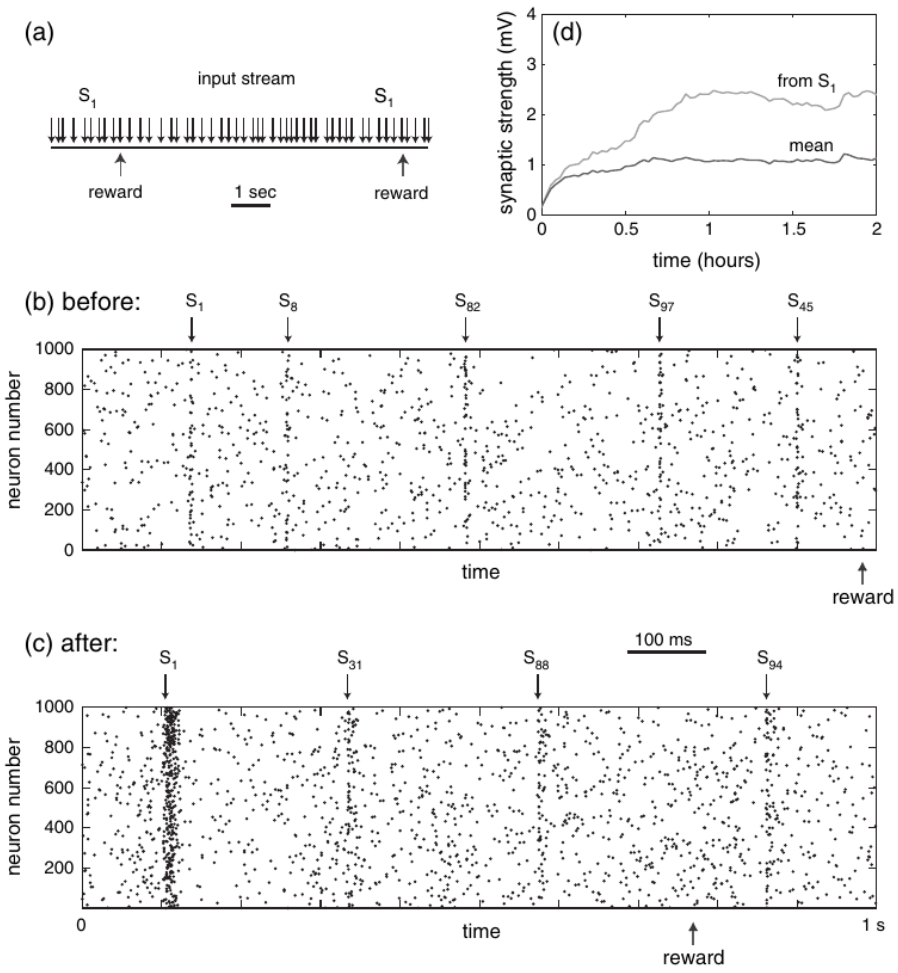
Figure 5 – An illustration of the MSTDPET learning rule, proposed by Florian (2007)



Source: Adapted from (FLORIAN, 2007)

50 neurons each are selected to represent 100 different stimuli to the network. For instance, stimuli S_1 causes a predefined set of 50 neurons to fire by injecting a superthreshold current for 1ms. A reward signal is provided with a random delay of up to 1s after stimuli S_1, as illustrated in Figure 6(a). Note that a random sequence of other stimuli is presented to the network between stimuli S_1 and the corresponding reward. Over time, the synapses connecting to the 50 neurons representing the set S_1 are strengthened significantly above the network's average (Figure 6(c) and (d)).

Figure 6 – An illustration of Pavlovian learning by a randomly connected network of Izhikevich neurons



Source: Adapted from (IZHIKEVICH, 2007)

3 LEARNING FROM SPARSE AND DELAYED REWARDS WITH A MULTI-LAYER SPIKING NEURAL NETWORK

This chapter was originally published as a manuscript at the International Joint Conference on Neural Networks (IJCNN 2020) (CHEVTCHENKO; LUDERMIR, 2020).

3.1 INTRODUCTION

While recent advancements in neuromorphic hardware allow energy efficient synthesis of spiking networks, the training of such networks remains an open problem. An analysis of related works, presented in Section 3.2, suggests that previously proposed spiking networks for RL do not scale well with an increased number of sensors. Furthermore, the plasticity is usually restricted to a single layer and linearly separable problems. The main goal of this work is to present and evaluate a novel spiking architecture, overcoming the limitations of previous models in sensory space scalability. In this work, we focus on reinforcement learning with sparse and delayed rewards. Our SNN is evaluated on classical reinforcement learning and control tasks and compared to two common RL algorithms: Q-learning and deep Q-network (DQN). The proposed architecture has four distinct layers and addresses the limitation of previous models in terms of scalability with input dimensions. Our model is intended for future implementations on hardware. As such, we follow a strategy similar to recent works (MOZAFARI et al., 2019; HAZAN et al., 2018) and use simplified neural and synaptic plasticity models, prioritizing compactness over biological realism.

3.2 RELATED WORK

Early versions of SNNs with reward-modulated plasticity are independently proposed by Izhikevich (2007) and Florian (2007). In both instances, eligibility traces and STDP are employed. In particular, Florian (2007) demonstrate that a fully connected multilayered network of spiking neurons can solve a temporally coded XOR problem with delayed reward.

Potjans, Diesmann e Morrison (2011) propose the first spiking network to implement an actor-critic framework, a classical RL algorithm. The network's functionality is demonstrated on a discrete grid-world environment, where the agent is able to move in four cardinal directions to reach a target position. Frémaux, Sprekeler e Gerstner (2013) also propose a continuous time actor-critic model with temporal difference (TD) error driven plasticity. The proposed neuromodulation is later included in a more general three-factor learning rule in Frémaux e Gerstner (2016). The network is evaluated on a larger grid-world task with obstacles, as well as two control tasks, including an acrobot. It should be noted that both Potjans, Diesmann e Morrison (2011) and Frémaux, Sprekeler e Gerstner

(2013) implemented a spiking network with no hidden layers. Thus, each neuron in the input layer, also called *place neurons*, is used to encode a specific state of the environment. For instance, in order to keep the acrobot problem computationally tractable and limit the number of input neurons, Frémaux, Sprekeler e Gerstner (2013) uses a custom encoding to reduce the dimensionality of the environment and limit the resolution per sensor. Despite this, the number of place neurons still reaches 11,025 for a four-dimensional state space. The present work proposes a novel architecture that does not significantly increase the number of neurons for higher dimensional problems. For instance, 60 input neurons is enough for an acrobot task with six-dimensional observations.

Nakano et al. (2015) tackle the input dimensionality problem by proposing an SNN version of a restricted Boltzmann machine (RBM) with memory. The network is able to navigate on a T-maze by using visual cues from 28×28 binary images. While this approach is successful on a high dimensional task, the network is previously trained for visual processing in a supervised manner. Thus it is unclear whether such an architecture could be applied to high dimensional control problems and continuous learning.

Rueckert et al. (2016) introduce an SNN with a model-based RL, including separate state and context neural populations. While this network is able to solve planning tasks, it is still limited to a few input dimensions. A related, but more scalable learning rule through imitation is introduced by Tanneberg et al. (2016). The present work draws inspiration from the models proposed by Rueckert et al. (2016) and Tanneberg et al. (2016), while maintaining the reinforcement learning paradigm and introducing a population coding that can be scaled well to more than four dimensions.

Wilson et al. (2017) evaluate different versions of R-STDP with randomly connected Izhikevich neurons, introducing variable dopamine concentration over time and space. The network is trained for locomotion of virtual aquatic animals and is distributed over the agent's body. Variable spatio-temporal dopamine concentration seems to significantly outperform standard R-STDP (IZHIKEVICH, 2007). The network is trained with a sparse reward signal, although continuous reward is also evaluated with less success. A trained network is able to correlate the input stimulus with output neurons in order to maximize motion speed. In order to induce movement of the animat's body, the input of the network is a set of fixed periodic signals, without sensory information.

Bing et al. (2018a) show that R-STDP outperforms other state-of-the-art algorithms based on SNNs for a line-following robotic task. A similar but multilayered SNN model is demonstrated on a virtual snake robot in (BING et al., 2019), outperforming a single-layer counterpart. In both works, the reward signal is continuously provided for correction and the learning process is not aimed at distal rewards, i.e. that are temporally distant from the actions that led to them.

The power consumption advantage of neuromorphic hardware is demonstrated by Wunderlich et al. (2019), where a small spiking network with R-STDP learns to play a

pong game on a BrainScaleS 2 neuromorphic system. As with the previous related works, the input layer is used to encode all of the system's states.

A synaptic plasticity rule with two types of synapses is proposed by Yuan et al. (2019). This rule is intended to better approximate the biological plasticity mechanisms and involves stochastic and deterministic synapses. While it is a novel learning rule, providing insight into how the brain performs RL, the scalability of the neural architecture is not addressed.

3.3 PROPOSED SPIKING NETWORK

A diagram of the proposed SNN is shown in Figure 7. In this setup there are three neurons per dimension ($n_d = 3$) and the agent chooses between four possible actions. For legibility, only the connections from active input neurons are shown. The illustrated environment is a 3×3 grid and the agent receives positional information from two sensors. The input layer contains $N_i = n_d \times n_s$ neurons, where a group of n_d neurons is used to encode signal intensity from each of n_s sensors in a one-hot configuration. This way, the input layer produces n_s spikes at any given timestep. Four discrete actions are possible, representing movement in each cardinal direction. Due to winner take all (WTA) activation, a single action neuron is selected each timestep to produce a spike. The following sections describe in more details the neural model and the structure of the proposed network.

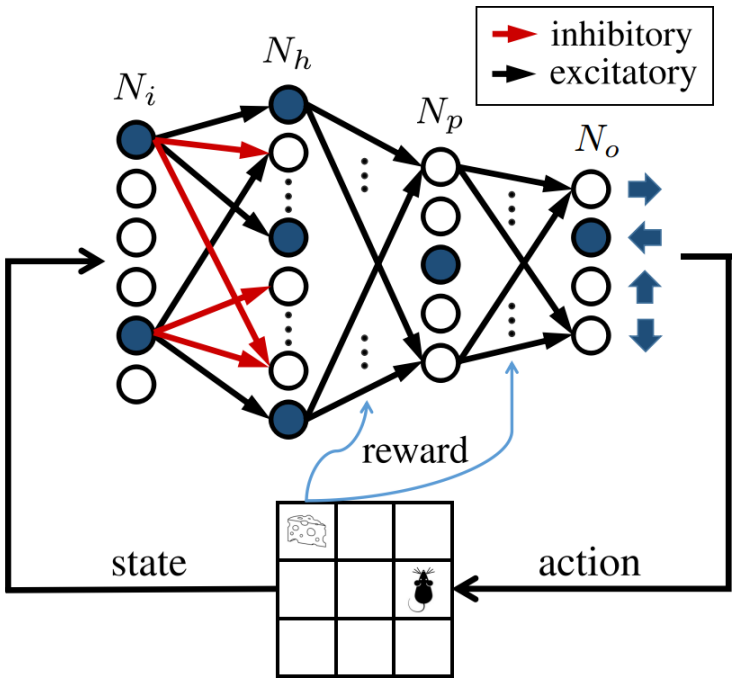
3.3.1 Neural Model

Except for the input layer, the neural model is a stochastic leaky integrate-and-fire (LIF). The neuron accumulates a charge from pre-synaptic spikes and emits a spike to post-synaptic neurons when the internal potential reaches a threshold. Throughout this thesis, j is used to refer to the pre-synaptic neuron and i is used for the post-synaptic one. A discrete-time update rule for the potential of the neuron i is

$$v_i(t) = v_i(t-1) - \frac{v_i(t-1)}{\tau_m} + \xi_i(t) + \sum_j w_{j,i} s_j(t-1), \tag{3.1}$$

where $v_i(t)$ is the potential of neuron i, τ_m is the membrane discharge time constant and $\xi_i(t)$ is the Gaussian noise. This noise is centered at zero with 0.2 standard deviation and applied only to neurons in *Place* and *Output* layers. This parameter was adjusted during initial experiments, where it was verified that very low noise level would make the network converge faster to a sub-optimal policy. On the other hand, a larger noise level would result in an overly random behavior and slower convergence. The term $w_{j,i}s_j(t-1)$ is the potential induced in neuron i from pre-synaptic spike by neuron j, weighted by the synaptic efficacy between neurons i and j.

Figure 7 – Diagram of the proposed spiking network. In this setup there are three neurons per dimension ($n_d = 3$) and the agent chooses between four possible actions. For legibility, only the connections from active input neurons are shown.



Source: The author, 2023

3.3.2 Synaptic Plasticity

The following is a simplified version of the reward-modulated STDP plasticity rule introduced by Florian (FLORIAN, 2007). When post-synaptic neuron i fires, the eligibility trace $Z_{j,i}$ is set to the value of the pre-synaptic trace P_{-}, decaying exponentially with a time constant τ_e:

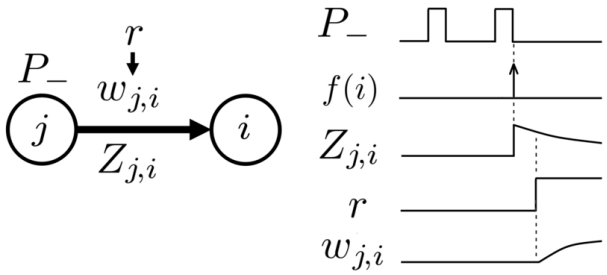
$$Z_{j,i}(t) = Z_{j,i}(t-1) - \frac{Z_{j,i}(t-1)}{\tau_e}. \tag{3.2}$$

The change in synaptic efficacy between two neurons depends on pre-synaptic and post-synaptic spike times, as well as on the reward signal

$$w_{j,i}(t) = w_{j,i}(t-1) - \frac{w_{j,i}(t-1)}{\tau_s} + \eta\ r(t)\ Z_{j,i}(t), \tag{3.3}$$

where τ_s is the synaptic tag discharge time constant and η is the learning rate. Note that both τ_e and τ_s are defined as larger than 1. The reward signal $r(t)$ is broadcasted to all plastic synapses. The rule in Equation 3.3 is illustrated in Figure 8. Presynaptic trace P_{-} is used to indicate that neuron j has produced a spike on the previous time step. Spiking of the postsynaptic neuron i triggers an update of the eligibility trace $Z_{j,i}$. Subsequent broadcast of the reward signal r prompts an update to the synaptic efficacy $w_{j,i}$. Note that the time constants τ_e and τ_s are adjusted according to the expected duration of an episode for a specific task.

Figure 8 – The synaptic plasticity model. Presynaptic trace P_- is used to indicate that neuron j has produced a spike on the previous time step. Spiking of the postsynaptic neuron i ($f(i)$) triggers an update of the eligibility trace $Z_{j,i}$. Subsequent broadcast of the reward signal r prompts an update to the synaptic efficacy $w_{j,i}$



Source: The author, 2023

3.3.3 Hidden Layer

The hidden layer allows the network to solve problems that are not linearly separable in the feature space. This layer contains N_h neurons. Each neuron in the input layer sends n_+ excitatory and n_- inhibitory synapses to randomly chosen neurons in the hidden layer. For example, in Figure 7 each input neuron sends two excitatory and two inhibitory synapses. In other words, input and hidden layers are sparsely connected with $N_i \times (n_+ + n_-)$ synapses, where n_+ and n_- are much smaller than N_h. In the experiments presented in this work the weights of these synapses are kept constant ($+1$ or -1) and are not subject to modulation by reward.

3.3.4 Place Neurons

This layer is inspired by the neuronal activity of the hippocampus and represents the position of the agent in the sensory space (O'KEEFE; NADEL, 1978). In some of the related works it is also the input layer and fully describes the state of the system. In the proposed model there are N_p neurons, which is much smaller than the number of all possible input states ($n_d^{n_s}$). Thus, unlike in previous works, place neurons do not encode all possible positions in the sensory space, but rather the ones that the agent has visited leading to a reward. An intuitive and experimentally supported (MAMAD et al., 2017) reasoning is that we expect an animal in a labyrinth to better remember a path that leads to successive rewards rather than other available routes. The hidden layer and place neurons are fully connected with plastic synapses, following the update rule from Equation 3.3, with presynaptic trace P_- resetting to $-n_+$. While a number of hidden neurons can produce a spike at each time step, only the place neuron with the highest potential sends a spike to the output layer. This is similar to lateral inhibition found in Frémaux, Sprekeler e Gerstner (2013) or a global max-pooling layer in Mozafari et al. (2019) and increases the

robustness of the learning process.

3.3.5 Output Layer

The output layer in the proposed architecture is very similar to previous works, such as Potjans, Diesmann e Morrison (2011), Frémaux e Gerstner (2016), Bing et al. (2018a) and has N_o neurons, each representing a possible discrete action. As in the place layer, in order to ensure a single action choice only the neuron with highest potential is allowed to spike at each time step.

The synapses coming to this layer are subject to the same learning rule as in the previous layer. An intuitive explanation is as follows. Suppose that the agent visits a number of states, performing an action choice at each state. This is translated to corresponding place and action neurons producing a spike. As described by Equation 3.2, the synapses between these neurons will be tagged by the eligibility trace $Z_{j,i}$ and gradually discharge at rate τ_e. When a state transition leads to a reward, the synaptic weight will be increased proportionally to the eligibility trace, following Equation 3.3. In other words, state-action choices that happened closer to the reward signal become more likely to occur in the future trials. Over time, more distant choices also become increasingly likely as the corresponding synaptic weights are successively increased.

Exploration/exploitation balance is also important for a successful learning. Typically the agent starts randomly exploring the environment until a reward is provided. Over time, as the plastic connections between hidden, place and action layers are reinforced, the choice of actions becomes more deterministic. Thus, a policy is learned and exploited by successive trials. A minimum random choice of action is used to ensure some level of exploration even after multiple rewarded trials. The experiments described in Section 3.4 use a constant 2% random action probability.

3.4 EXPERIMENTS

3.4.1 Setup

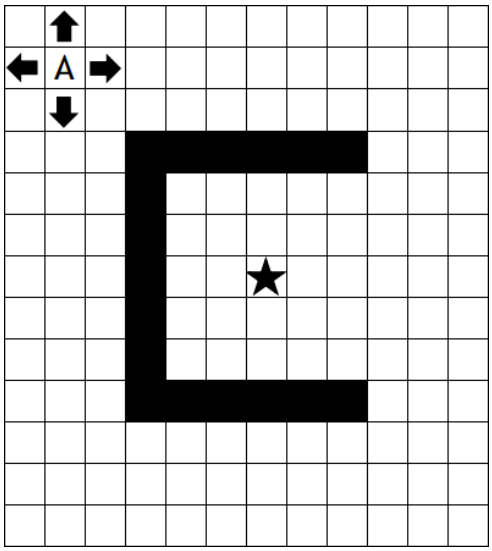
The proposed model is compared against baseline algorithms on two simulated tasks: maze and acrobot. Both environments provide a sparse and binary reward upon successfully reaching the goal state, but require different capabilities from the agent. A detailed description of these environments is provided below.

3.4.1.1 Maze task

The Morris water-maze is a classical neuroscience experiment, in which a mouse swims in a pool with opaque water, looking for a submerged platform. Variations of this task are found in several related works, such as (POTJANS; DIESMANN; MORRISON, 2011) and

(FRÉMAUX; SPREKELER; GERSTNER, 2013). This environment is illustrated in Figure 9. Only the state in the center is rewarded (+1). The agent (A) starts in the upper-left corner and can move in four cardinal directions. Collisions with the U-shaped wall or with the borders of the maze are not penalized but will make the agent return to the previous position. In order to learn the task, an agent has to perform a long sequence of discrete actions and reach a goal that comprises 0.04% of all possible states in a 50×50 maze. A binary reward is received upon reaching the target position.

Figure 9 – Illustration of the maze task. Only the state in the center is rewarded (+1). The agent (A) starts in the upper-left corner and can move in four cardinal directions. Collisions with the U-shaped wall or with the borders of the maze are not penalized but will make the agent return to the previous position



Source: The author, 2023

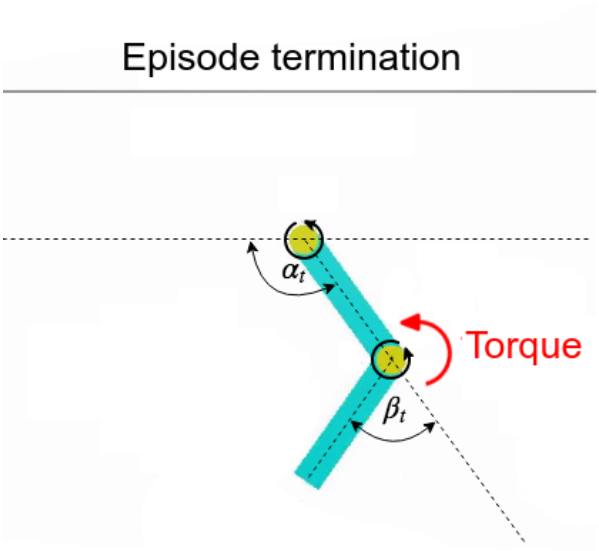
Preliminary experiments were also conducted with variations of this task, such as different maze sizes, no walls and up to 5 dimensions. The proposed SNN performed well in these exploratory experiments and the limiting factor seemed to be the probability of randomly finding the target, which becomes exponentially harder as the number of dimensions is increased. As described in the following section, the *acrobot* environment is more suitable to evaluate scalability with higher dimensional observations.

3.4.1.2 Acrobot

In order to demonstrate control over a higher and more dynamic state space, we turn to another classical problem in RL literature. The setup is illustrated in Figure 10 and the goal is to lift the tip of the robot to a certain level. The environment provides sine and cosine of both joint angles, as well as the respective angular velocities, making a total of six state dimensions. The second joint is weakly actuated and the system includes

gravitational pull. To solve this task, the agent has to consistently swing the actuated joint, building up the energy. In contrast with the maze, the acrobot problem does not have a single target state, but the search is performed over a higher, six-dimensional state space. In order to encourage policies that reduce the time it takes to reach the target state, the agent receives a binary reward signal when the target is reached with lower latency than the last 100 trials. This environment is equivalent to the one provided by OpenAI Gym (BROCKMAN et al., 2016), except for the reward policy, which is modified to be binary and sparse.

Figure 10 – Illustration of the acrobot task



Source: The author, 2023

3.4.2 Baseline Models

As seen in Section 3.2, existing spiking architectures would require an impractical number of state neurons for control tasks with more than four sensors. Thus, the proposed architecture is compared to Q-learning (WATKINS, 1989), a classical RL algorithm, as well as its more recent variation, Deep Q Network (DQN) (MNIH et al., 2015), both described in Sections 3.4.2.1 and 3.4.2.2. It is worth noting that the present work is not intended as an advance in general reinforcement learning. A competitive comparison between spiking and state-of-the-art neural networks for RL is left for future works and should include a multiobjective analysis of performance as well as memory and power requirements.

3.4.2.1 Q-learning

This is a classical model-free RL algorithm, proposed by Watkins (1989), itself derived from temporal difference (TD) learning (SUTTON, 1988). An agent tries an action a_t at a

state s_t, following a transition to a new state s_{t+1} and a reward r. By trying actions in different states, the agent eventually can learn which state-action tuples are more likely to lead to a future reward. The estimation of this likelihood is stored in a Q-table $Q(s_t, a_t)$. The update is performed after the transition to the state s_{t+1} as follows:

$$Q(s_t, a_t) = Q(s_t, a_t) + \eta(r + \gamma Q_{\max}(s_{t+1}) - Q(s_t, a_t)), \tag{3.4}$$

where η is the learning rate, γ is the discount factor and $Q_{\max}(s_{t+1})$ is the maximum Q-value at the state s_{t+1}. The Q-table is randomly initialized with a uniform distribution between -1 and 1. To induce exploration, the selected action is not always determined by its value:

$$a_t = \begin{cases} \arg\max_i Q(s_t, a_i), & \text{with probability } \epsilon \\ \textit{random}, & \text{with probability } 1 - \epsilon, \end{cases} \tag{3.5}$$

where ϵ is the exploration probability. A common strategy is to start learning with a high value of ϵ and gradually decrease it to a small baseline.

3.4.2.2 DQN

The traditional Q-learning algorithm works well with a limited number of states and actions, but as demonstrated in Section 3.4.4, becomes inefficient for problems with multiple dimensions. One common solution is to replace a Q-table, representing all possible states and actions, with a parameterized function $\boldsymbol{\theta}$, such as a neural network: $Q(s_t, a_t; \boldsymbol{\theta}_t)$. Then, the update rule in the Equation 3.4 becomes:

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta}_t + \eta(Y_t^Q - Q(s_t, a_t; \boldsymbol{\theta}_t))\nabla_{\boldsymbol{\theta}_t} Q(s_t, a_t; \boldsymbol{\theta}_t) \tag{3.6}$$

$$Y_t^Q = r + \gamma Q_{\max}(s_{t+1}; \boldsymbol{\theta}_t), \tag{3.7}$$

where Y_t^Q is a target value for $Q(s_t, a_t; \boldsymbol{\theta}_t)$, representing expected value from taking action a_t on state s_t.

The deep Q network (DQN) algorithm, proposed by Mnih et al. (2015), introduces two new techniques that significantly improve learning: periodic target network update and experience replay. This way the target value Y_t^Q is computed with an offline copy of the parameters $\boldsymbol{\theta}_t$, which is updated every τ_u steps. The online neural network is trained through gradient descent using batches, sampled from a large memory bank of observed (s_t, a_t, r, s_{t+1}) transitions.

It is worth pointing to a distinction between Q-learning and the DQN algorithms. While both Q-learning and the proposed spiking network learn from discrete inputs, DQN uses a multilayer perceptron to approximate the Q-function. On the maze task, this means

Table 1 – Q-learning hyperparameters search space

Hyperparameter	**Search Space**
Learning rate η	[0.01, 0.1, 0.2, 0.5]
Discount factor γ	[0.5, 0.8, 0.9, 0.95, 0.99]
Exploration fraction	[0.01, 0.1, 0.2, 0.5]

Source: The author, 2023

that DQN would receive as an input the XY coordinates of the agent. A discrete one-hot encoding could also be used, as with the proposed spiking architecture. However, the latter approach greatly increases the number of input neurons, slowing down the learning process significantly. On the other hand, when using two neurons as an input for XY coordinates, a large maze would require pinpoint accuracy for the agent not to miss the relatively small target area (0.04% of a 50×50 maze). In both encoding methods, albeit for different reasons, DQN was found to struggle to find a path in the maze environment larger than 10×10. Because Q-learning is a classical baseline for low dimensional tasks, DQN was used as a baseline for the acrobot environment only, where it outperforms Q-learning.

3.4.3 Hyperparameters

Hyperparameter selection is an important part of training a model and usually has significant influence on performance. A grid search over all possible combinations of parameters is often impractical due to the computational cost of a single simulation. Manual tuning is a viable alternative but it is often not better than performing random trials (BERGSTRA et al., 2011). Thus, in order to ensure a fair comparison, the hyperparameters of the baseline algorithms are optimized. Note that the proposed spiking network is adjusted manually and optimization is left for future works.

Considering the large number of parameters and the high computational cost per evaluation, automatic hyperparameter search is often preferred. In the present work we employ a commonly used Tree-of-Parzen-Estimators (TPE) algorithm, introduced by Bergstra et al. (2011). The hyperparameters are iteratively evaluated and optimized to minimize a cost function. For both the maze and acrobot tasks the cost function is the area under the latency curve of a trial (see Figures 11 and 13).

The hyperparameter search spaces for the Q-learning and DQN algorithms are described in Tables 1 and 2. The exploration fraction refers to the portion of the total number of trials that the exploration factor takes to decrease linearly from initial 100% to a baseline of 2%. The search space is based on values commonly found in related literature, as well as on preliminary experiments.

The optimization algorithm is executed for a minimum of 100 trials and the evaluation

Table 2 – DQN hyperparameters search space.

Hyperparameter	**Search Space**
Learning rate η	[1e-6, 1e-5, 1e-4, 1e-3]
Discount factor γ	[0.9, 0.95, 0.99, 1.0]
Exploration fraction	[0.1, 0.2, 0.5]
Activation function	[relu, sigmoid, tanh]
Number of hidden layers	[1, 2]
Neurons per hidden layer	[16, 32, 64, 128, 256]
Optimizer	[GD, Adam]
Replay buffer size	[1000, 10000, 50000]
Mini-batch size	[16, 32, 64, 128, 256]
Update period τ_a (steps)	[500, 1000, 2000]

Source: The author, 2023

is terminated when no better configuration is proposed in the last 50 iterations. Each optimization trial contains 1,000 episodes for the maze task and 2,000 episodes for the acrobot task. The final configurations for each algorithm are listed below.

Q-learning is used as a baseline for both maze and acrobot tasks:

- Learning rate η – maze/acrobot: 0.2
- Discount factor γ – maze: 0.95, acrobot: 0.9
- Exploration fraction – maze/acrobot: 0.1

The DQN algorithm is optimized on the acrobot task. We also evaluate the default parameters provided by OpenAI Baselines (DHARIWAL et al., 2017), shown in parentheses:

- Learning rate η – 1e-3 (5e-4)
- Discount factor γ – 0.95 (1.0)
- Exploration fraction – 0.1
- Activation function – relu (tanh)
- Number of hidden layers – 2 (1)
- Neurons per hidden layer – 256 (64)
- Optimizer – Adam
- Replay buffer size – 50000
- Mini-batch size – 256 (32)

Table 3 – Hyperparameters of the proposed SNN

Hyperparameter	**Maze**	**Acrobot**
# of input neurons N_i	100	60
# of hidden neurons N_h	300	300
# of place neurons N_p	300	500
# of output neurons N_o	4	3
Input-hidden n_+ and n_-	15	15
Hidden-place τ_e	65	10^2
Hidden-place τ_s	10^6	10^5
Place-action τ_e	20	65
Place-action τ_s	10^5	10^4

Source: The author, 2023

- Update period τ_a – 500 (1000)

The manually adjusted parameters of the proposed SNN are presented in Table 3.

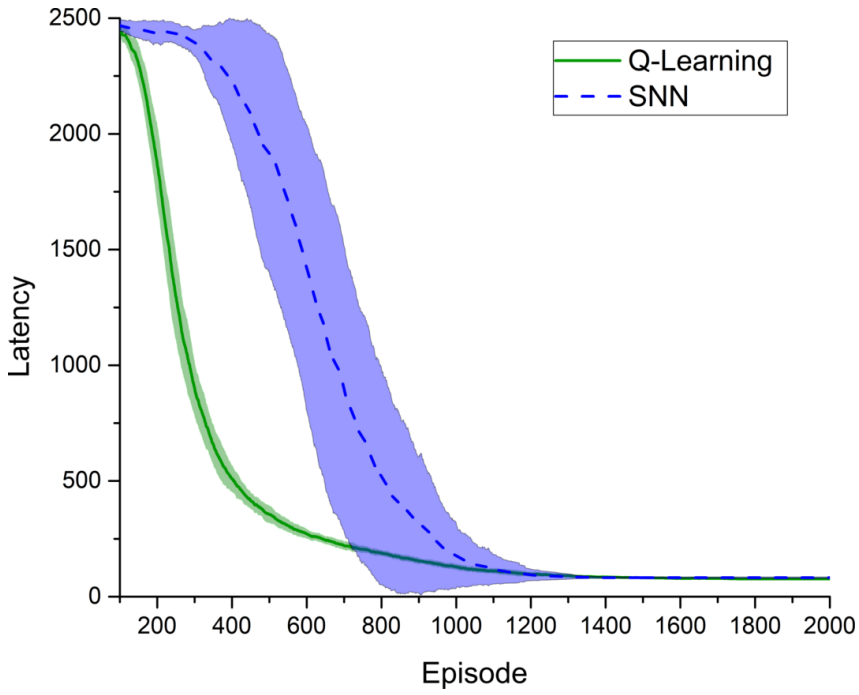
3.4.4 Results

The proposed spiking network and the Q-learning algorithm are evaluated on the 50×50 maze task, described in Section 3.4.1.1. We measure the latency of each agent — the number of steps it takes to reach the target state. Figure 11 shows the average latency as a function of episodes. Shaded region is the standard deviation over 10 trials. The latency is averaged continuously with a running window of 100 episodes. An episode is terminated if the agent reaches the goal state or performs over 2,500 actions. A trial comprises the training of a single agent over 2,000 episodes and the presented curves are an average of 10 trials.

As seen in Figure 11, the agent controlled by our multilayer SNN takes longer than Q-learning to start exploiting a policy, although both reach a stable latency at about 1,300 episodes. This result contrasts with previous works, where single layer spiking networks achieved similar latency decay to TD learning (POTJANS; DIESMANN; MORRISON, 2011) (FRÉMAUX; SPREKELER; GERSTNER, 2013). A possible explanation is that while the single layer networks implement a spiking version of TD learning, our network follows a different approach and relies on synaptic traces for temporal back-propagation.

Additionally, the proposed network has more layers and synapses that are subject to modulation by reward. While this slows down learning, the spiking network is ultimately able to learn the necessary series of actions to increase reward incidence and reduce latency. A sample path is illustrated in Figure 12. Color codes correspond to episodes from the same trial.

Figure 11 – Average latency on the maze task. Shaded region is the standard deviation over 10 trials



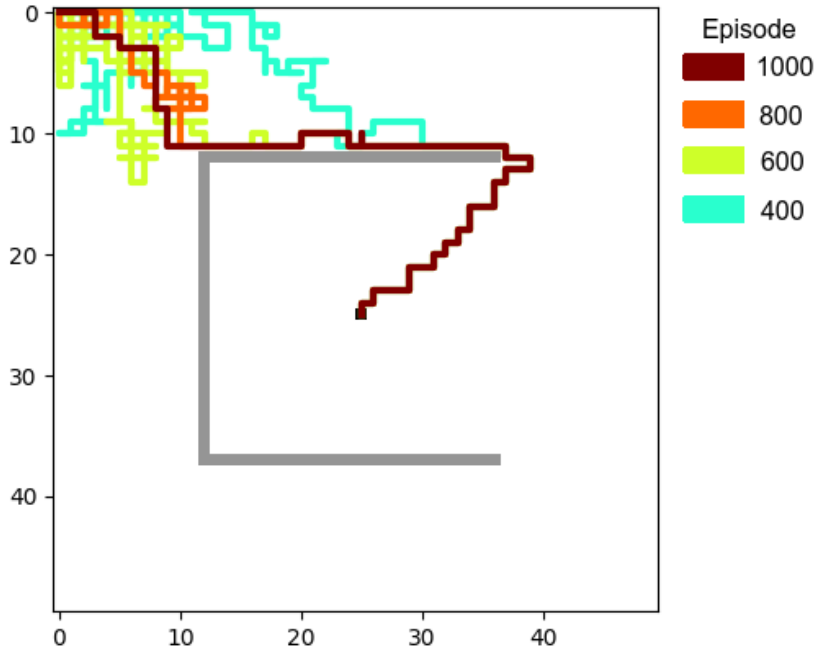
Source: The author, 2023

We now evaluate the proposed SNN, Q-learning and the deep Q network on the acrobot problem. This task requires accurate control over a 6 dimensional observation space, as described in Section 3.4.1.2. While DQN networks are trained on continuous input from the six sensors, both Q-learning and SNN receive a discrete input with 10 levels for each sensor, i.e. a 10^6 state space. The discrete control output is the same for all agents — apply left, right or no torque.

Despite this being a more challenging task, our SNN agent is able to learn a control strategy, as shown in Figure 13. Shaded region is the standard deviation over 10 trials. Standard deviation is high for the default DQN and is not shown for readability. As in the maze task, the presented latency is averaged over 100 episodes. The termination of a single episode is achieved after 500 steps or when the acrobot is able to reach the target height, illustrated in Figure 10. As this is a more difficult task, a trial lasts for 10,000 episodes.

An analysis of Figure 13 shows that Q-learning is unable to find efficient control strategies. This is expected, as on this task the Q-table has 3×10^6 entries, accounting for all possible state-action combinations. Executing the algorithm for significantly longer (100,000 episodes) also does not improve the average latency. On the other hand, the proposed SNN is able to bring the average latency down to about 300 steps. An example of a successful episode is shown in Figure 14. The agent reaches the goal height in the

Figure 12 – Sample paths followed by an SNN on the maze task. Color codes correspond to episodes from the same trial.

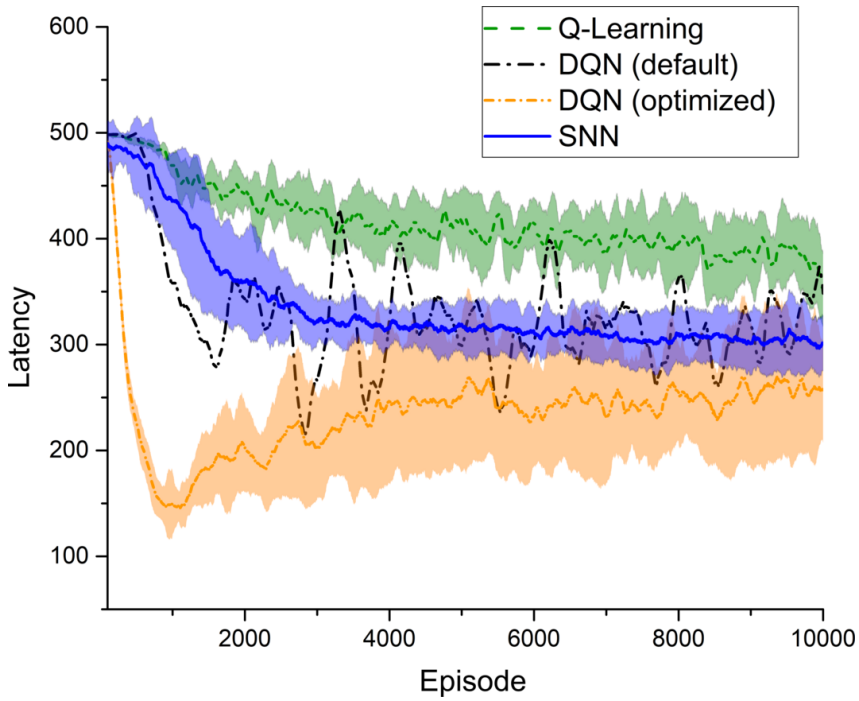


Source: The author, 2023

final step. A video sample is available at <https://youtu.be/r5BZMZ4hb_o>.

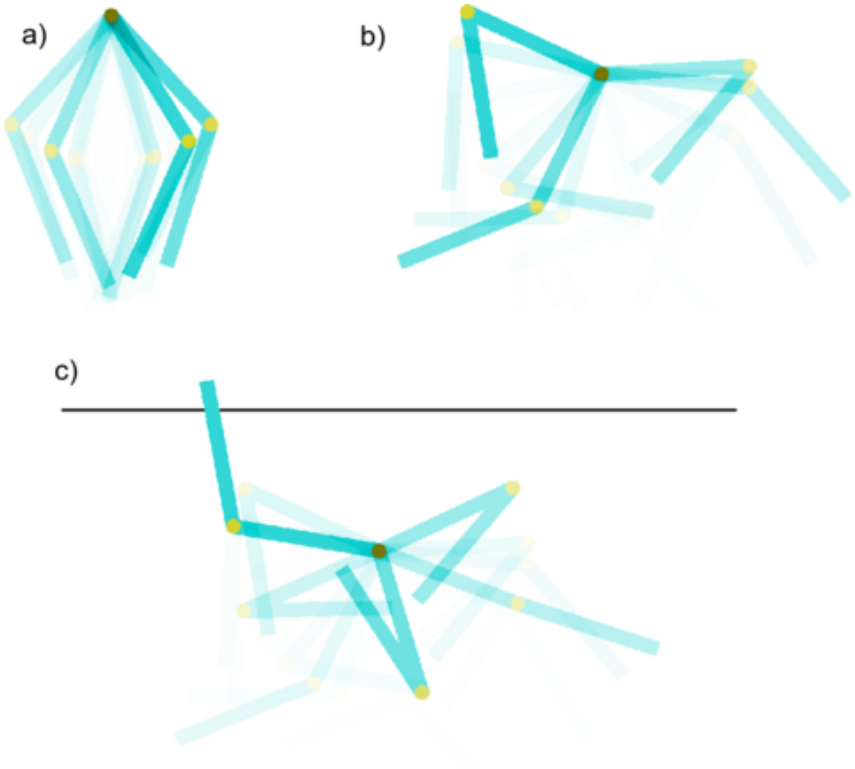
Note that both Q-learning and the SNN perform control over the same discrete space. Surprisingly, the default configuration of DQN struggles to learn a stable controller, although at times it can reach better solutions on average than the SNN agent. The DQN with optimized hyperparameters is more stable than default version and learns faster than other agents, although it is still less stable than the spiking network.

Figure 13 – Average latency on the acrobot task. Shaded region is the standard deviation over 10 trials. Standard deviation is high for the default DQN and is not shown for readability.



Source: The author, 2023

Figure 14 – Time-lapse of a successful episode with the SNN controller. The agent reaches the goal height in the final step



Source: The author, 2023

3.5 CONCLUSION

The present work advances the field by presenting a new spiking architecture that overcomes a limitation of previous models in terms of sensory space scalability. Additionally, the proposed network is aimed for implementation on neuromorphic hardware by using sparse connectivity and simplified neural and plasticity models.

Experimental evaluation shows that our SNN outperforms traditional Q-learning on an acrobot task with six-dimensional observations, using 60 input and 800 hidden neurons. Such task would require 10^6 input neurons in the previously proposed models (POTJANS; DIESMANN; MORRISON, 2011; FRÉMAUX; SPREKELER; GERSTNER, 2013; FRÉMAUX; GERSTNER, 2016; WILSON et al., 2017). While an optimized deep Q-network is able to find better solutions on this task, the spiking controller is more stable and does not require a large memory bank. This makes it an efficient solution particularly for systems with memory and energy constraints. It is also worth noting that our network is trained through simplified STDP instead of gradient descent, which improves potential energy requirements (WUNDERLICH et al., 2019).

4 COMBINING STDP AND BINARY NETWORKS FOR REINFORCEMENT LEARNING FROM IMAGES AND SPARSE REWARDS

This chapter was originally published as a manuscript at Neural Networks (CHEVTCHENKO; LUDERMIR, 2021).

4.1 INTRODUCTION

In the previous chapter, we introduced a simple network that used a discrete binary vector with sparse activations for input encoding. For instance, in the case of the acrobot environment, we utilized six input dimensions, encoding them using ten distinct bins. This ensured that precisely 10% of the input neurons are active at any given timestep. However, this approach has its limitations when dealing with denser input signals.

This chapter explores the impact of connectivity and synaptic parameters between the input and hidden layers on the network's capacity to learn a policy from more complex inputs. To achieve this, we employ a pre-trained binary Convolutional Neural Network (CNN) to extract features from naturalistic images. The derived dense feature vector is then integrated with an SNN, which is trained online through reward-modulated STDP.

The spiking network is an extension of its previous version, with improvements in architecture and dynamics to address a more challenging task. We focus on extensive experimental evaluation of the proposed model with optimized state-of-the-art baselines, namely proximal policy optimization (PPO) and deep Q network (DQN). The models are compared on a grid-world environment with high dimensional observations, consisting of RGB images with up to 256×256 pixels. The experimental results show that the proposed architecture can be a competitive alternative to deep reinforcement learning (DRL) in the evaluated environment and provide a foundation for more complex future applications of spiking networks.

The remaining of this chapter is organized as follows. Related works are presented in Section 4.2. Section 4.3 provides a detailed description of the proposed model. Section 4.4 discusses the experimental setup and presents the results of our study. Finally, Section 4.5 concludes the chapter and outlines potential avenues for future research.

4.2 RELATED WORKS

The problem of training SNNs from images with delayed rewards is previously addressed by Nakano et al. (2015) with a spiking version of a free-energy-based RL model (OTSUKA; YOSHIMOTO; DOYA, 2010). A T-maze environment is used to validate the proposed model with visual cues consisting of 28×28 binary images from the MNIST dataset. The present work is evaluated on a more challenging version of this task, using RGB images with up

to 256×256 pixels. Inspired by Nakano et al. (2015), we also evaluate a pre-trained binary CNN as a feature extractor. However, this network is trained on a different set of images from the ones presented in the testing environment.

Wunderlich et al. (2019) make a small spiking network with R-STDP learn to play a simplified version of an Atari pong game on a BrainScaleS 2 neuromorphic system. The main objective of this work is to demonstrate the power consumption advantage of neuromorphic hardware and the presented architecture has similar scalability limitations of previous works.

Kaiser et al. (2019) propose a framework for neural simulation that interacts with robotic models. A previously proposed synaptic plasticity rule SPORE (KAPPEL et al., 2018) is evaluated on ball balancing and line following tasks, with observations provided by a small number of visual neurons (16×16 and 16×4 respectively). The authors show that gradually decreasing learning rate over time improves the performance of the algorithm. This work provides an important contribution to joining the fields of robotics and spiking networks. However, the framework is aimed at simulating biologically realistic neurons and synapses, without demonstrating competitive performance in comparison with deep learning methods. While biological realism is not an obvious impediment to state-of-the-art performance, in the present work we choose to focus on comparing the proposed model with modern DRL algorithms (MNIH et al., 2015; SCHULMAN et al., 2017).

A hybrid approach to training SNNs is recently explored by Tang, Kumar e Michmizos (2020). In this work, training of a spiking actor is assisted by a deep learning critic, using the DDPG algorithm (LILLICRAP et al., 2015). Thus, when a critic network successfully approximates the reward function of the environment, the spiking actor can be efficiently deployed on a robot. The joint training is optimized and shown to be more effective on the test environment than DDPG alone and DDPG-to-spiking conversion methods. On the other hand, another recent work by Bing et al. (2020) compares a spiking network, trained via R-STDP, to DQN-to-spiking conversion. On the evaluated lane following task, the R-STDP is shown to be more effective than DQN converted to a spiking network after training. Similarly, in the current work we combine a pre-trained binary CNN with a spiking model trained via R-STDP and show it to be competitive against PPO and DQN algorithms.

A learning rule called e-prop is proposed by Bellec et al. (2020) and applied to a recurrent spiking network. This rule is shown to approximate the performance of an LSTM network trained via backpropagation through time on two discrete Atari games. This is a promising method and it is worth noting that the spiking recurrent network is trained online, receiving only the current frame at each step. However, the results are not directly compared to DRL alternatives and speed of learning is not addressed. Similarly to the present work, the temporal parameters related to eligibility traces are gradually increased during simulation. The current work is focused on demonstrating competitive

performance on both speed and quality of learning, comparing our approach to optimized state-of-the-art RL algorithms. In the future we intend to explore recurrent connectivity as well as online training of a convolutional spiking network, both present in Bellec et al. (2020).

Independently from Bellec et al. (2020), Chung e Kozma (2020) propose an STDP-based learning rule for a spiking network with feedback modulation. The learning rule is based on rate of firing instead of individual spikes. While this network is shown to be able to solve two classical control problems, the performance is comparable to an online actor-critic agent from Sutton e Barto (2018). In a previous work (CHEVTCHENKO; LUDERMIR, 2020), we have shown that the proposed spiking model can achieve comparable performance to the DQN algorithm on a similar control task to the one found in Chung e Kozma (2020). However, an optimized DQN is shown to find better control policies than the spiking model. Thus, we dedicate special attention to optimization of hyperparameters for baseline models in this chapter, as this can have a significant impact on performance (ENGSTROM et al., 2019).

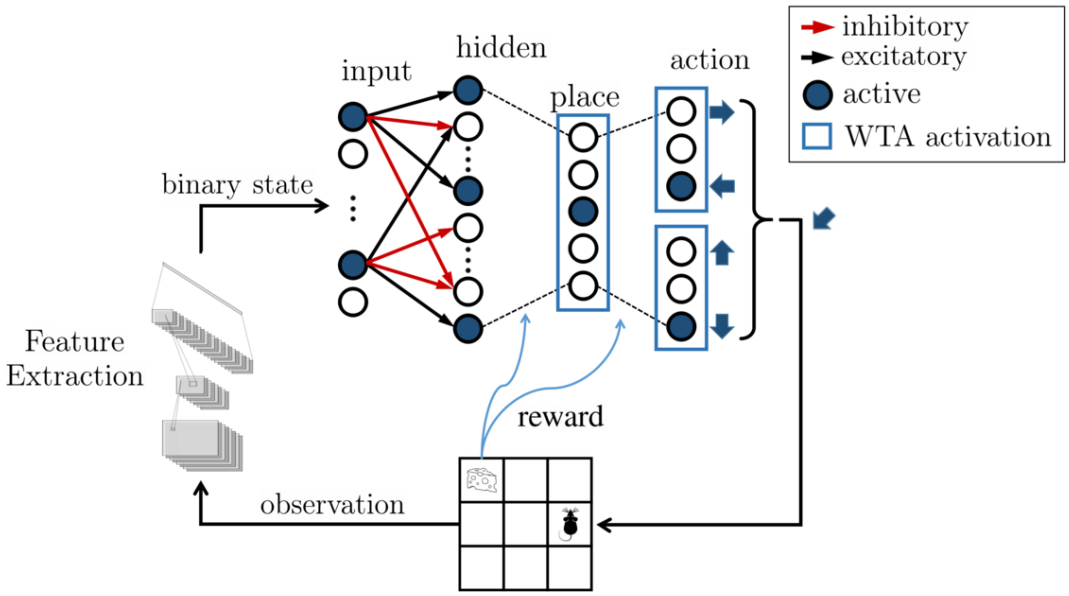
4.3 THE PROPOSED NETWORK

The proposed model is based on a previously introduced architecture (CHEVTCHENKO; LUDERMIR, 2020). The main components of the network and its interaction with the environment are illustrated in Figure 15. Some active neurons are highlighted in this illustration. Observations from the environment are processed by a feature extraction network. Input and hidden layers are sparsely connected with static inhibitory and excitatory synapses, while the next layers are fully connected with plastic synapses. A winner-take-all (WTA) activation is used in the place and action layers. The agent-environment interaction is done in a cyclic manner. At each step, the environment is modified by the agent's action and provides a new observation, as well as a scalar reward signal when appropriate. The following sections describe in more details the structure and dynamics of the model.

4.3.1 Neural model

The spiking model used in this work is analogous to the integrate-and-fire (IF) model, without leakage and refractory period. The internal potential of the neuron is a weighted sum of presynaptic spikes with added noise. If the internal potential reaches a certain threshold, the neuron emits a single spike and resets it's potential. Our model is simulated with a custom code instead of an existing software for spiking neurons. Thus, by prioritizing ease of simulation and implementation on hardware, we do not use biologically realistic parameters. Consequently, the parameters presented in this work are unitless, although a physical dimension can be assigned.

Figure 15 – Diagram of the proposed model. Some active neurons are highlighted in this illustration. Observations from the environment are processed by a feature extraction network. Input and hidden layers are sparsely connected with static inhibitory and excitatory synapses, while the next layers are fully connected with plastic synapses. A winner-take-all (WTA) activation is used in the place and action layers



Source: The author, 2023

The neuron accumulates a charge from presynaptic spikes and emits a spike to postsynaptic neurons when the internal potential reaches a threshold. Throughout this chapter, j and i are used to refer to the presynaptic and postsynaptic neurons, respectively. A discrete-time update rule of neuron i is as follows:

$$v_i(t) = v_i(t-1) + \xi_i(t, \sigma_n) + \sum_j w_{j,i} s_j(t), \tag{4.1}$$

where $v_i(t)$ is the potential of neuron i and $\xi_i(t)$ is a Gaussian noise. The noise is centered at zero with σ_n standard deviation and applied only to neurons in *place* and *output* layers. The activation of the presynaptic neuron is a binary variable s_j. The term $w_{j,i}s_j(t-1)$ is the potential induced in neuron i from presynaptic spike by neuron j, weighted by the synaptic efficacy between neurons i and j. Other layer-specific parameters are described in the next sections.

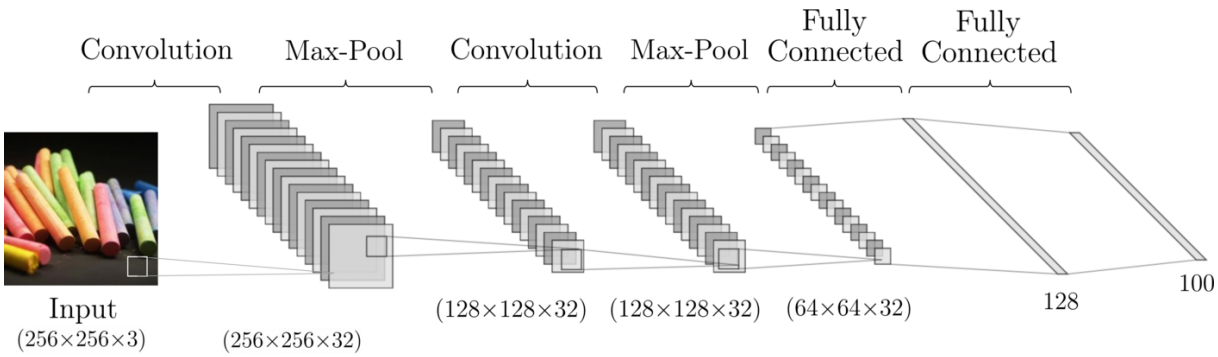
4.3.2 Feature extraction network

In the previous work, the spiking network is shown to learn RL tasks from sparse inputs with up to six sensors in a one-hot discretization method (CHEVTCHENKO; LUDERMIR, 2020). While this encoding is effective for a small number of analog sensors, it results

in an impractically large input layer when dealing with higher dimensional inputs, such as images. For instance, a 128×128 RBG image with a 10-bit one-hot encoding would require 491,520 input neurons. In order to overcome this limitation, a convolutional neural network is used as a feature extractor. A binary CNN (COURBARIAUX et al., 2016) is evaluated in the present work. Note that with an appropriate binarization method, any feature extractor could be used in this step. However, a binary network offers two appealing advantages when combined with an SNN. First, a pre-trained binary CNN can be at least as power-efficient as a spiking network. Second, any layer of a binary network can simply become the input layer of an SNN, where '1' indicates a spike. In the present implementation, a '-1' activation is considered an absence of a spike in the corresponding input neuron. Thus, no analog-to-digital conversion is required, as would be the case with a standard CNN, resulting in a larger input layer or time-based encoding. It is worth noting that a custom hardware would be required to simultaneously exploit benefits of both spiking and binary networks. The hybrid model presented in this chapter could allow a more compact implementation of the stateless feature extraction and input-hidden neurons, where synaptic traces are not used.

An illustration of the binary convolutional architecture used in this work is provided in Figure 16. Differently from a traditional supervised learning scenario, this network is trained on a small number of randomly chosen images. This speeds up training, as the network does not have to learn how to categorize different images into classes. Instead, the convolutional filters are trained to discriminate between a random set of 100 images. The hidden fully connected layer with 128 neurons is used as an input for the spiking network. Note that the feature extraction network is trained on a separate class of images from what is used as observations in the RL environment. More details on the dataset partition are provided in Section 4.4.1.

Figure 16 – A diagram of the binary CNN used as a feature extractor. The network is pre-trained on a small set of randomly selected images



Source: The author, 2023

The training of a binary network as a feature extractor can be summarized in the

following steps:

1. Select a random subset of 100 images for training.
2. Create a binary network with 100 output neurons.
3. Train the network as a classification problem with 100 categories until convergence.
4. Save the trained model, discarding the last layer.

The implementation of this network is largely based on the one provided in the Fast Machine Learning Lab repository (KREIS; HOANG; DUARTE, 2020) and is included in the sample code (See A for supplementary materials). An Adam optimizer is used with default parameters. The model is trained during 1000 epochs and the learning rate decays linearly from 10^{-3} at the start of training to 10^{-4} at the end.

The trained feature extractor has 128 output neurons, each binary neuron producing a +1 or -1 activation. In the present work a -1 is considered as an absence of a spike in the input layer of the SNN. It is also possible to use a pair of inhibitory and excitatory neurons in the input layer. In this case, a -1 or +1 activation would produce a spike in the corresponding inhibitory or excitatory neurons. However, this approach is not evaluated in the experiments as it would double the number of neurons in the input layer.

4.3.3 Input and hidden layers

These first two layers are functionally similar to layers in a fully connected binary neural network and have static weights. The purpose of the hidden layer is to allow the network to solve problems that would not be linearly separable in the feature space, mapping the input signal to a sparse representation with a reduced overlap. A biological inspiration behind this type of connectivity can be found, for instance, in a drosophila mushroom body (ZHANG et al., 2013). More specifically, a sparse connectivity between the antennal lobe (AL) and the mushroom body (MB) is shown to decrease overlap between similar signals and improve discrimination capability (see Section 2.6 in Zhang et al. (2013) for more details).

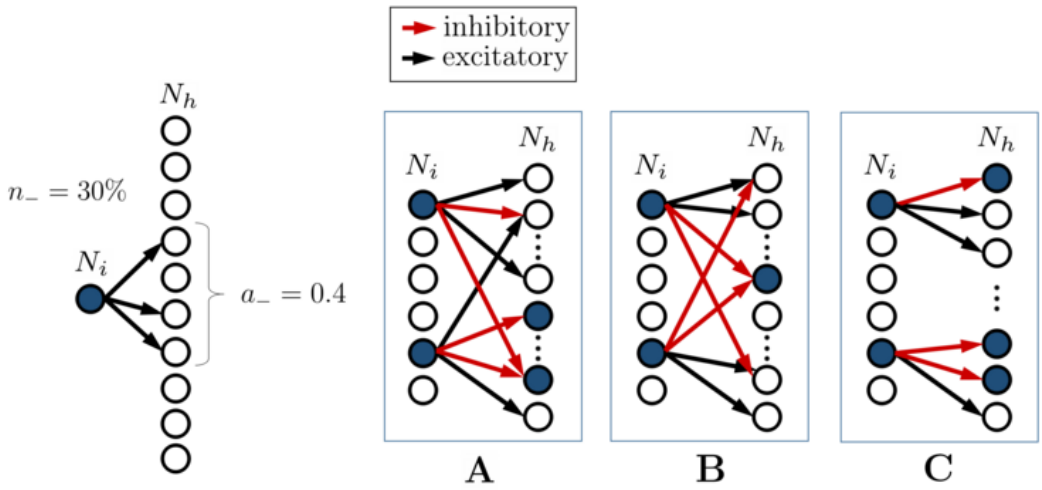
In the present work, we evaluate how different connectivity patterns affect the network's ability to learn, i.e. to produce a sequence of actions that increase the probability of reaching a rewarded state. Details on these experiments are given in Section 4.4.3.2. Notably, a simple random and sparse connectivity that was used in the previous work is not the best option when the number of input neurons is increased. See Section 4.4.4.3 for experimental comparison between different connectivity schemes.

Considering a network with N_i and N_h input and hidden neurons, the connectivity from each input to multiple hidden neurons is parametrized in the following manner:

- n_+ and n_- – number of excitatory and inhibitory synapses, expressed as a percentage of N_h.
- a_+ and a_- – amplitude of connectivity. A value close to 0 means that the input neuron will make connections only to closest neurons in the hidden layer. A value of 1 corresponds to random connectivity to any neuron in the next layer.

Excitatory and inhibitory synapses have a static weight of +1 and -1, respectively. A sample of connectivity patterns that can be encoded by the above parameters is presented in Figure 17. Note that the input pattern produced by the feature extraction network is relatively dense, with about 50% of input neurons active at any time step. In this case, a sparse hidden layer is also used to reduce overlap between these binary patterns. Sample activations from input and hidden layers are presented in Figure 19. Additional experiments, comparing the overlap of spiking patterns from input and hidden layers, are presented in Section A.

Figure 17 – On the left: illustration of inhibitory connectivity for a single input and 10 hidden neurons. Excitatory connectivity is defined analogously. On the right, a sample of possible input-hidden connectivity patterns. **(A)** completely random, **(B)** local excitatory and global inhibitory connections and **(C)** local inhibitory and excitatory connections. Filled circles indicate active neurons



Source: The author, 2023

4.3.4 Place neurons

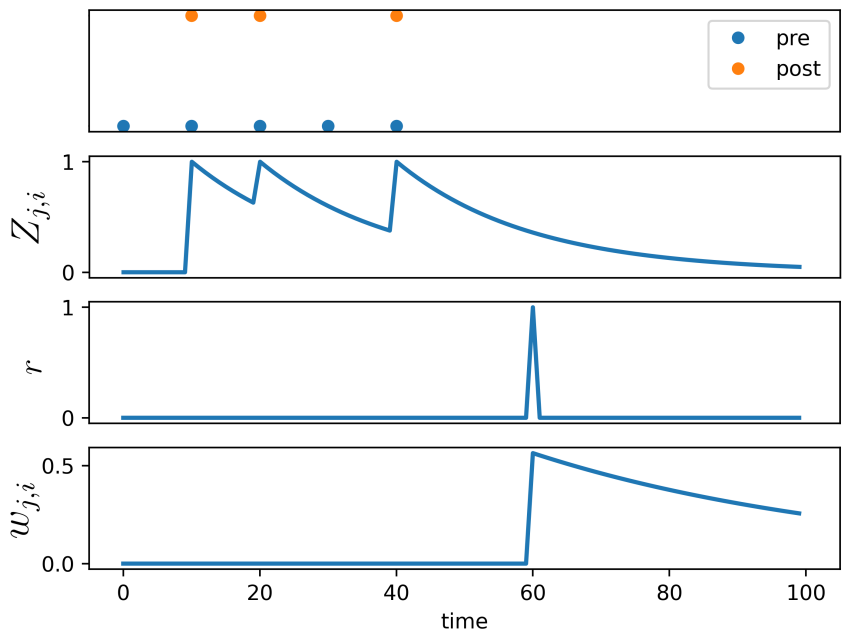
This layer mimics the experimentally observed behavior of biological place cells within the hippocampus (O'KEEFE; NADEL, 1978). These cells tend to fire in groups associated with

the position of the agent in the sensory space. Moreover, place cells are demonstrated to fire consistently along a path that leads to a reward (MAMAD et al., 2017). In the present work, place cells are fully connected to the previous hidden layer with plastic synapses. Intuitively, this layer is used to reduce the dimensionality of the task to the most frequently visited states.

The synaptic plasticity used here is a simplified version of the reward-modulated STDP rule introduced by Florian (2007). A binary variable P_- is used to indicate the state of the presynaptic neuron from the hidden layer. P_- is set to 1 when a neuron has fired and -1 otherwise. When the postsynaptic neuron i fires, the eligibility trace $Z_{j,i}$ is incremented by P_-, decaying exponentially with a time parameter τ_e, as defined in the Equation 3.2.

The change in synaptic strength between two neurons depends on the eligibility trace, modulated by a global reward signal, as per Equation 3.3. The reward signal $r(t)$ is broadcasted to all plastic layers and reinforces synapses with larger trace values ($Z_{j,i}$). In the absence of a reward, all weights decay towards zero and consequently encourage a random exploratory behavior. This is also referred as L2 regularization in related literature (CHUNG; KOZMA, 2020). The above plasticity is illustrated in Figure 18. In this example, τ_e is set to 20 and τ_s to 50.

Figure 18 – Illustration of the synaptic plasticity model. A spike by the postsynaptic neuron triggers an update of the eligibility trace $Z_{j,i}$. A later broadcast of the reward signal r prompts an update to the synaptic efficacy $w_{j,i}$



Source: The author, 2023

The time parameters τ_e and τ_s can also be dynamically adjusted, according to the duration of a trial and reward frequency of a specific task. In the present work τ_s is kept at a constant value, and τ_e increased linearly during a trial from 1 to a maximum value τ_s^{max}. The hyperparameters used during the experiment are optimized and presented in Section 4.4.3.2. A possible alternative would be to use a range of time constants on a larger network, which may be the case in biological brains (GERSTNER et al., 2018).

The potential of a place neuron, described by Equation 4.1, is clipped between -1 and +1. When a spike is produced, the corresponding neuron is reset to the minimum potential of -1. This is done to ensure that a different place cell is active at each time step.

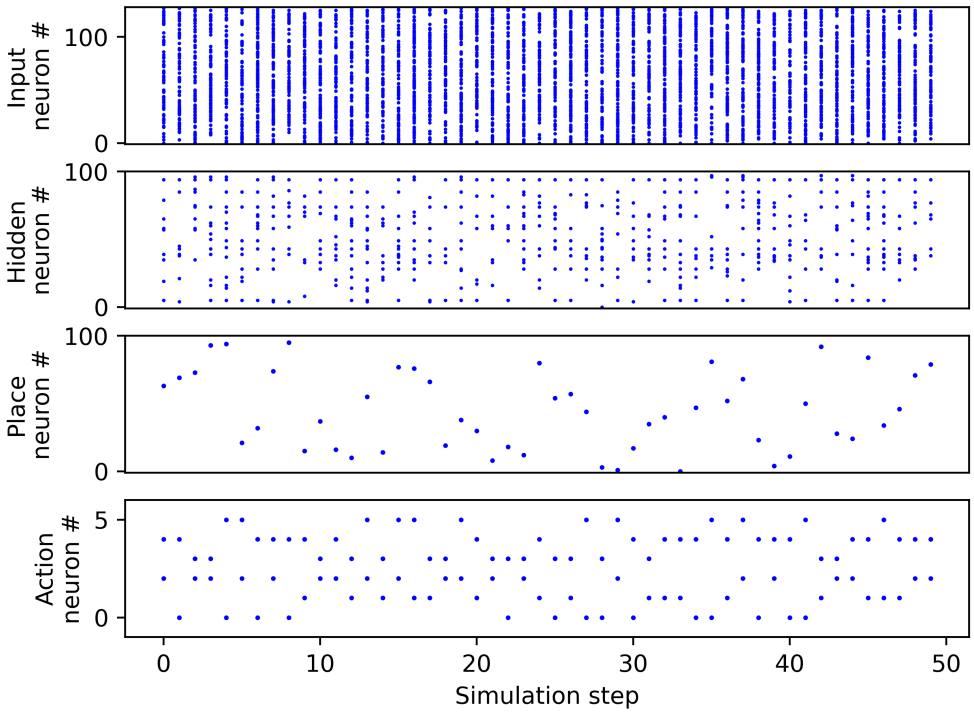
While a number of hidden neurons can produce a spike at each time step, the winner-take-all (WTA) activation is used in the place layer, i.e. only the place neuron with the highest potential sends a spike to the next layer. This is similar to lateral inhibition found in Frémaux, Sprekeler e Gerstner (2013) or a global max-pooling layer in Mozafari et al. (2019) and increases the robustness and speed of the learning process.

4.3.5 Output layer

In contrast to the previous version of this network (CHEVTCHENKO; LUDERMIR, 2020), the number of neurons in the output layer scales linearly with the number of output dimensions. Thus, the action layer has N_a neurons, divided in the same number of groups as there are action dimensions. For example, as illustrated in Figure 15, an agent in a 2D maze can move independently along two axes with three discreet movements available for each axis. Note that, as in the *place* layer, a neuron with the highest potential in a group is going to produce a spike. In order to increase exploration in the beginning of a trial, the noise in the *output* layer is linearly decreased from σ_n^{max} to a final σ_n value.

Figure 19 illustrates sample activation from input to output layers for 50 discrete steps. Note that without the reward signal to drive the synaptic plasticity, the activations are random.

Figure 19 – Sample of activation from input to the output layers for 50 discrete steps. Each dot corresponds to an active neuron at the time step indicated by the horizontal axis.



Source: The author, 2023

4.4 EXPERIMENTAL EVALUATION

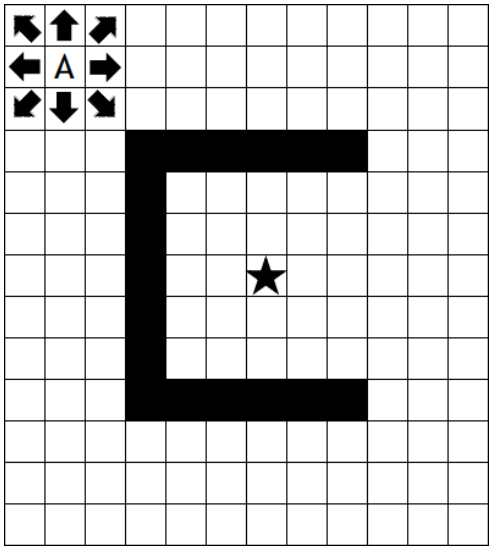
The proposed architecture is compared against two state-of-the-art deep reinforcement learning algorithms: PPO (SCHULMAN et al., 2017) and DQN (HASSELT; GUEZ; SILVER, 2016). Both baseline algorithms are evaluated with optimized hyperparameters to ensure a fair comparison. A detailed description of the experimental setup is provided below.

4.4.1 Environment

While the architecture presented in Section 4.3 can be applied to arbitrary RL tasks with a discrete and finite environment, we focus on a commonly used grid-world environment for benchmarking. This environment is inspired by a classical neuroscience experiment called Morris water-maze, in which a mouse swims in a pool with opaque water, looking for a submerged platform. The grid-world is illustrated in Figure 20. Analogous environments, albeit with low dimensional observations, can be found in related works (CHEVTCHENKO; LUDERMIR, 2020; POTJANS; DIESMANN; MORRISON, 2011; FRÉMAUX; SPREKELER; GERSTNER, 2013; ROSENFELD; SIMEONE; RAJENDRAN, 2019). Additional

experiment in Section A provides a comparison between this environment with image and positional observations.

Figure 20 – Illustration of the grid environment. The agent (A) starts in a random corner and can move in 8 discrete directions. Collisions with the U-shaped wall and with the borders of the maze are considered, but not penalized. Only the state in the center is rewarded (+1). Note that observations provided by the environment are not positions, but colored images from a dataset, illustrated in Figure 21



Source: The author, 2023

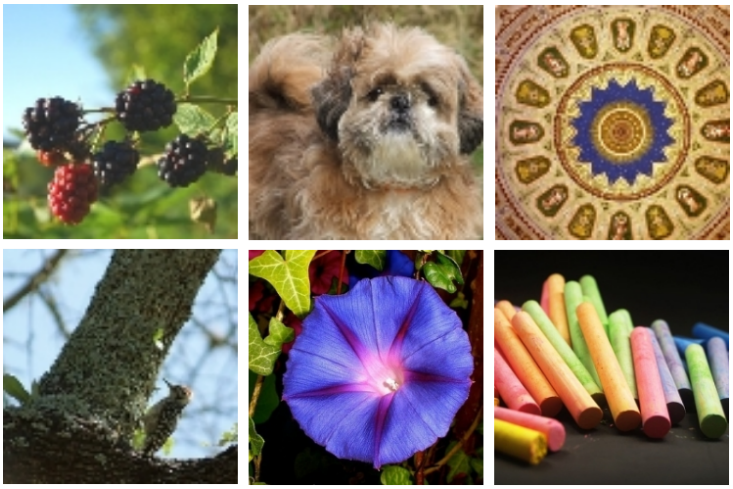
The grid-world is adapted to evaluate the ability to learn from higher dimensional observations. To this end, the position of the agent in the grid corresponds to a color image of up to 256×256 pixels (CHALADZE; KALATOZISHVILI, 2017). Each image is uniquely and randomly assigned. A sample of the dataset is provided in Figure 21. Thus, this task becomes a more challenging version of a digit matching T-maze problem, considered by Nakano et al. (2015). In order to consistently receive a reward, the agent has to take a long series of discrete actions, based on color image observations.

The Linnaeus 5 dataset (CHALADZE; KALATOZISHVILI, 2017), illustrate in Figure 21, contains 5 classes: berry, bird, dog, flower and other (negative set). We select 100 random images from the negative subset for training of the binary CNN, described in Section 4.3.2.

4.4.2 Baseline models

Here we briefly describe two DRL algorithms that are used as baselines. Both are well known and have been extensively benchmarked. Our implementation is based on the Stable Baselines3 (SB3) library, which is a set of reliable implementations of reinforcement

Figure 21 – A sample of images used as multidimensional observations for the agent



Source: The author, 2023

learning algorithms in PyTorch (RAFFIN et al., 2019). No modifications are made to the algorithms, except to allow hyperparameter tuning, as described in Section 4.4.3.

4.4.2.1 Deep Q network

This can be viewed as an extension of a classical Q-learning algorithm, proposed by Watkins (1989), itself derived from temporal difference (TD) learning (SUTTON, 1988). The original Q-learning algorithm is suitable for discrete actions but is inefficient on multi-dimensional observations, as previously demonstrated on an *acrobot* problem (CHEVTCHENKO; LUDERMIR, 2020).

This limitation of a tabular Q-learning can be mitigated by replacing a table of Q values with a function approximator, such as a neural network. Furthermore, the deep Q network (DQN) algorithm, proposed by Mnih et al. (2015), introduces two new techniques that significantly improve learning:

- Periodic *target* network update – in order to improve the stability of the learning process, there are two copies of the policy network. A *target* network is used for policy execution, while the *online* network is modified during training. The *target* network is periodically updated from this copy.
- Experience replay – the *online* network is trained through gradient descent using batches, sampled from a large memory bank of observed transitions.

Both the network update period and the size of the training batch are among the hyperparameters that can significantly influence the performance of a DQN. These and other hyperparameters are selected through optimization, as described in Section 4.4.3.

Note that when observations consist of images, or similar multidimensional inputs, the memory bank required to successfully train an agent can become prohibitively large.

4.4.2.2 Proximal policy optimization

Proposed by Schulman et al. (2017), this algorithm is currently one of the most popular and best performing on a range of applications. Compared to DQN, PPO typically requires a much smaller memory buffer and the training is computationally less expensive. PPO is an on-policy algorithm, i.e. the same policy network is modified during learning and is used to make decisions. In contrast, the DQN algorithm uses one network to interact with the environment while another one is trained from the collected observations. One of the main contributions of PPO is that the update process is regulated in order to avoid making abrupt changes to the policy. This smooth on-policy update is used to mitigate the smaller memory and avoid committing to an unrecoverable policy. Even so, PPO can be notably brittle and usually requires some adjustments to achieve stable learning (ENGSTROM et al., 2019). In the present work we use an actor-critic policy network with a shared CNN for image processing. As with the DQN implementation, the architecture of the networks and the algorithm's hyperparameters are optimized for the benchmarking environment.

4.4.3 Hyperparameter optimization

Both baseline algorithms, as well as the proposed SNN, have a number of hyperparameters that can significantly influence speed and quality of learning in a given environment. While it is possible to perform manual tuning of parameters for better performance, this can often be no better than a random search (BERGSTRA et al., 2011). On the other hand, the computational costs of ANN training and environment simulation tend to make an exhaustive search in the hyperparameter space impractical. A popular middle ground is found in Bayesian optimization, which is used when there is a limited evaluation budget, such as deep learning. Thus, in order to avoid manually introducing bias into benchmarking models and ensure a fair comparison, we perform automatic hyperparameter optimization using a recently introduced Optuna framework (AKIBA et al., 2019).

4.4.3.1 Baseline algorithms

The following set of hyperparameters are found to more significantly impact the performance and are automatically optimized. These hyperparameters are selected based on the ones evaluated in related literature, as well as preliminary experiments with the grid-world environment and a range of models provided by SB3.

- Number of filters in the convolutional layers – a two-layered convolutional network is used for feature extraction.

- Number of neurons in the hidden layer – the third layer in the policy network is fully connected and acts as a hidden layer.
- Buffer size – this is the size of the memory used for policy training. As described in Section 4.4.2, DQN requires a significantly larger memory buffer than PPO.
- Learning rate – this parameter is evaluated in an order of magnitude range around the default value for the Adam optimizer.
- Batch size – as with most deep learning models, the size of the minibatch used for training is usually much smaller than the entire memory buffer.
- Discount factor – this parameter determines how much weight the model gives to events from a more distant past. A value of 0 means that the agent is trained based only on an immediate outcome. On the other hand, setting this value to 1 would indicate that all past iterations are considered to be equally important to the present outcome.
- Train frequency – this applies to DQN and represents the number of steps between training of the policy network. A value of 1 means that the weights of the network are adjusted at each simulation step, which results in additional computational cost.
- Exploration fraction – in order to encourage exploration of the environment, the initial policy of a DQN agent is completely random. The probability of taking a random action is over time decreased to a minimum value, which is also a hyperparameter.
- Number of surrogate loss optimization epochs – this parameter describes the number of epochs that the PPO policy is updated for.
- Clipping – as described in Section 4.4.2.2, this parameter is used by the PPO algorithm to adjust the level of possible change in the network in order to avoid unrecoverable policies. A larger value can increase learning speed, although at some stability cost.

The optimization criterion used in this work is to minimize the average latency measured during 1000 episodes, with the environment randomly initialized before each episode. This encourages both learning speed and low final latency. An optimization run for a given model consists of 100 iterations and the model with the lowest average latency is then used for experiments presented in Section 4.4.4. Models are optimized for two environment configurations: 10×10 and 20×20 grids, both with 64×64 RGB observations. The considered hyperparameters and the corresponding search spaces are presented on Tables 4 and 5 for the DQN and PPO algorithms respectively. The final selected hyperparameters are indicated on the tables with letters **a** (10×10 grid) and **b** (20×20 grid). Note that the search space includes the default parameters provided by Raffin et al. (2019).

Table 4 – Optimized hyperparameters of DQN

Hyperparameter	Search space
# of filters in the 1st convolutional layers	[16, 32(**a**), 64(**b**)]
# of filters in the 2nd convolutional layer	[16, 32(**b**), 64(**a**)]
# of neurons in the hidden layer	[64, 128(**a**,**b**), 256]
Buffer size	[10k, 20k(**a**), 50k(**b**)]
Learning rate	[10^{-3}, 10^{-4}(**b**), 10^{-5}(**a**)]
Batch size	[16, 32(**a**,**b**), 64]
Discount factor	[0.9(**a**,**b**), 0.99, 0.999]
Train frequency (steps)	[1(**a**,**b**), 5, 10]
Exploration fraction	[5%, 10%(**b**), 20%(**a**)]
Final exploration rate	[2.5%, 5%(**a**), 10%(**b**)]

Source: The author, 2023

Table 5 – Optimized hyperparameters of PPO

Hyperparameter	Search space
# of filters in the 1st convolutional layer	[16(**a**), 32, 64(**b**)]
# of filters in the 2nd convolutional layer	[16, 32(**a**,**b**), 64]
# of neurons in the hidden layer	[64, 128, 256(**a**,**b**)]
Buffer size	[1k(**a**), 2k(**b**), 4k]
Learning rate	[10^{-3}, 10^{-4}(**a**,**b**), 10^{-5}]
Batch size	[32, 64(**a**), 128(**b**)]
Discount factor	[0.9(**a**), 0.99(**b**), 0.999]
# of surrogate loss optimization epochs	[5, 10(**a**), 20(**b**)]
Clipping parameter	[0.1, 0.2(**b**), 0.4(**a**)]

Source: The author, 2023

While the baseline algorithms use a CNN policy network, we also evaluate a combination of a BinaryNet feature extractor and the PPO algorithm with and MLP policy. These experiment is described in Section A.

4.4.3.2 The proposed network

The optimization of the proposed network is divided into two parts. The connectivity between input and hidden layers is optimized in a linear track environment, inspired by a similar experiment described by Frémaux, Sprekeler e Gerstner (2013). The linear track is a simplification of the grid-world, described in Section 4.4.1. Due to its low simulation cost, it allows for a complete search in a small hyperparameter space.

In the linear track experiment, the agent is presented with a sequence of n observations (RGB images). Since the plastic weights of the network are initially set to zero, the agent

Table 6 – Connectivity hyperparameters

Hyperparameter	**Search space**
n_+	[5%, 10%, 20%, 50%]
n_-	[5%, 10%, 20%, 50%]
a_+	[0.1, 0.25, 0.5, 1.0]
a_-	[0.1, 0.25, 0.5, 1.0]

Source: The author, 2023

performs a series of random actions $a_1, a_2, ..., a_n$ due to a low noise level in each neuron. Next, a reward signal is broadcasted and the same sequence of observations is presented to the agent, which produces a new set of actions $a'_1, a'_2, ..., a'_n$. After the reward, the agent is expected to perform a similar sequence of actions, when presented with the same observations. Note that due to the noise, the actions are not entirely deterministic. Thus, an error rate E is computed by comparing the action pairs produced by the agent during the first and second runs:

$$E = \frac{1}{n}\sum_{i}^{n}(a_i == a'_i) \tag{4.2}$$

A low error rate means that the agent is able to successfully form synapses based on eligibility traces in order to reproduce the action sequence that have lead to a reward. A total of 256 combinations of input-hidden connectivity parameters is evaluated. The range of these parameters is presented in Table 6. For each combination of $[n_+, n_-, a_+, a_-]$ parameters, the error rate is an average of ten independent runs.

Other hyperparameters of the network are presented in Table 7. These are optimized in the same manner as the baseline algorithms and the selected hyperparameters are also indicated with letters **a** (10×10 grid) and **b** (20×20 grid). The parameter 'σ_n decrease time' indicates the # of timesteps σ_n takes to go from σ_n^{max} to minimum constant of 0.1. Likewise, 'τ_e increase time' is the simulation time required for τ_e to go from a minimum value of 1 to a maximum τ_e^{max}. The number of neurons in the hidden layer is kept constant at 1000 for all following experiments, except in Section 4.4.3, where the impact of individual parameters is evaluated.

4.4.4 Results

In the following experiments we measure the latency of each agent, i.e. the number of states it takes to reach the target. The latency is averaged with a running window of 10 episodes. An episode is terminated if the agent reaches the goal state or iterates over 100 (10×10 grid) or 400 (20×20 grid) steps. A trial comprises the training of a single agent over 1,000 episodes and the presented curves are an average of 10 trials.

Table 7 – Optimized hyperparameters of the proposed network

Hyperparameter	Search space
# of neurons in the *place* layer	[50, 100(**a**), 200(**b**)]
σ_n (place neurons)	[0.2(**b**), 0.4(**a**), 0.8]
σ_n^{max} (action neurons)	[0.2, 0.4(**a,b**), 0.8]
σ_n decrease time (action neurons)	[10^3, 10^4(**a,b**), 10^5]
τ_e^{max} (hidden-place layer)	[10, 20, 40(**a,b**)]
τ_e^{max} (place-action layer)	[10(**a,b**), 20, 40]
τ_e increase time (hidden-place layer)	[10^3, 10^4(**b**), 10^5(**a**)]
τ_e increase time (place-action layer)	[10^3, 10^4(**b**), 10^5(**a**)]
τ_s (hidden-place layer)	[1000, 2000(**a**), 4000(**b**)]
τ_s (place-action layer)	[1000(**a**), 2000, 4000(**b**)]

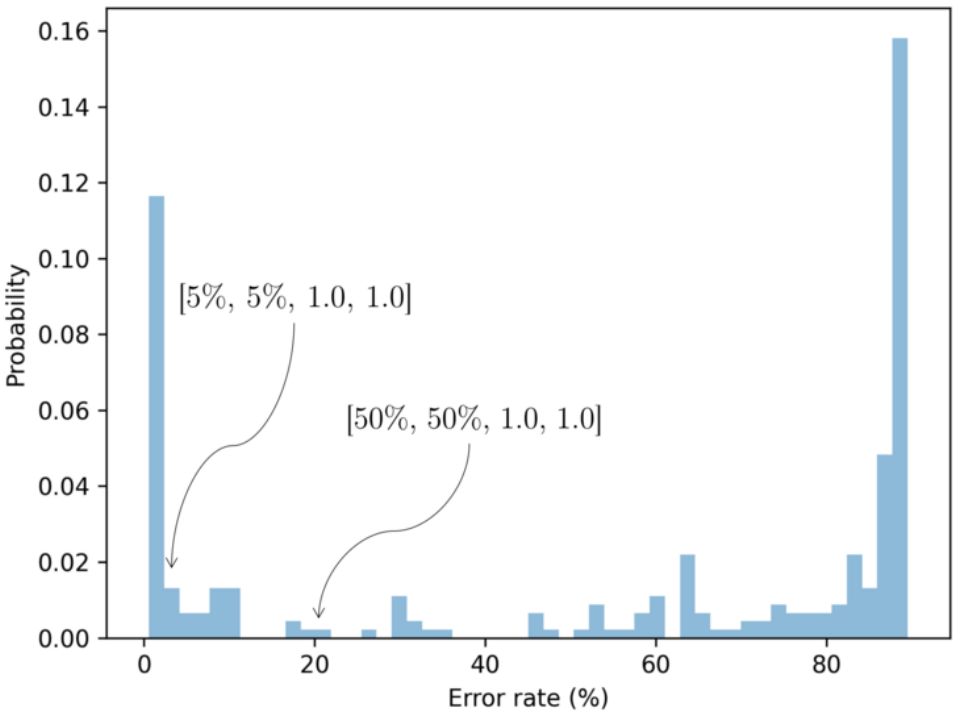
Source: The author, 2023

4.4.4.1 Connectivity optimization

The list of hyperparameters presented in Table 6 is exhaustively evaluated on a linear track environment, as described in Section 4.4.3. The error rate from Equation 4.2 is averaged for ten independent runs, with the track size set to 500 observations. Thus, for each of the ten runs, a unique random image is assigned to a step of the track. A distribution of the obtained results, obtained through grid search optimization of the input-hidden connectivity, is presented in Figure 22. This distribution suggests that a random connectivity is unlikely to produce a good results. The following connectivity parameters obtained the best average error rate of 0.5% and are selected for further evaluation: $[n_+, n_-, a_+, a_-]$=[5%, 10%, 1.0, 0.25]. A list with other ten best connectivity configurations is provided in Table 14. A sample of activations from input and hidden layers is also illustrated in Figure 47. Additional experiments in Section 4.4.4.3 suggest that the amplitude parameters a_+ and a_- do not play a significant role in the evaluated environment.

Besides this set of optimal parameters, Figure 22 also highlights the performances of two other configurations: a) the connectivity that was manually selected in the previous work (CHEVTCHENKO; LUDERMIR, 2020) [5%, 5%, 1.0, 1.0] and b) a more densely connected version [50%, 50%, 1.0, 1.0]. Also note that while configuration [5%, 5%, 1.0, 1.0] and the optimal one are relatively close in terms of evaluated error rate (0.5% vs 2%), there is a significant performance difference between the two networks, as evaluated in the grid-world environment (see Figure 26 for a qualitative comparison).

Figure 22 – Probability distribution of the error rate, obtained through grid search optimization of the input-hidden connectivity



Source: The author, 2023

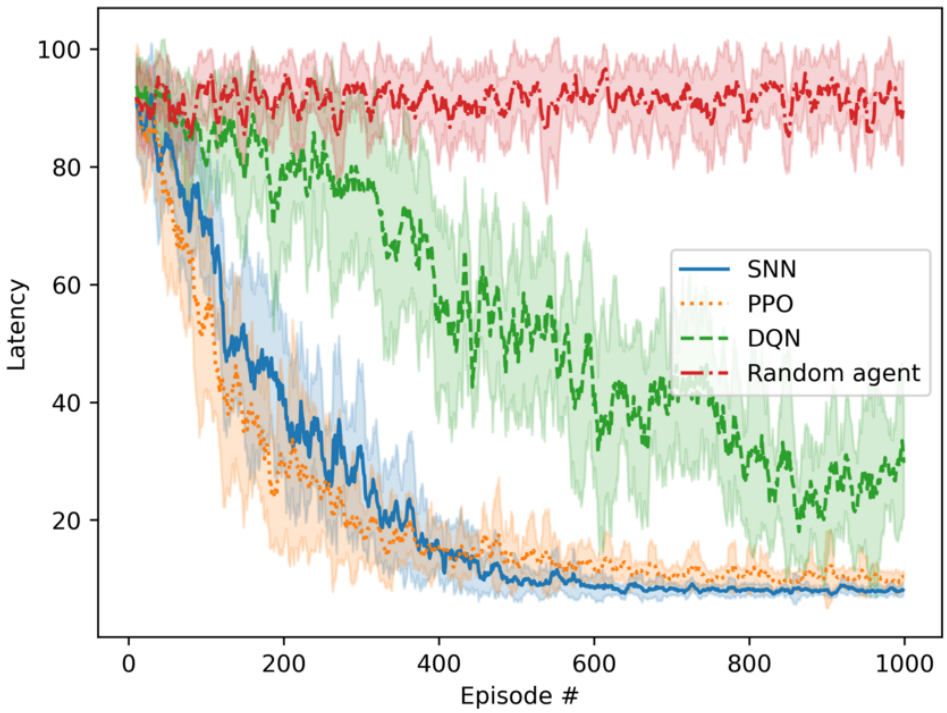
4.4.4.2 Comparison with baseline models

In this section we compare the proposed SNN with the baseline algorithms. The optimized set of hyperparameters for each algorithm is highlighted in Tables 4, 5 and 7.

SNN, PPO and DQN are evaluated on a grid-world of size 10×10 and observations consisting of 64×64 RGB images from the Linnaeus 5 dataset. The results are presented in Figure 23. Latency is averaged across ten trials with shaded regions corresponding to standard deviation. The performance of a random agent is also provided for reference. Both PPO and SNN significantly outperform the DQN algorithm. PPO and SNN have similar performance, however SNN achieves slightly better final latency.

Figure 24 provides results on a larger, 20×20 grid-world. The relative performance is similar to the 10×10 configuration, with SNN and PPO significantly outperforming DQN. Note that the optimized PPO achieves lower latency during first 200 episodes. However, the performance of SNN can be tailored to improve latency by adjusting a single temporal parameter, as demonstrated in Section 4.4.4.3.

Figure 23 – A comparison of the proposed network with PPO and DQN algorithms on a 10×10 grid-world. The observations consist of 64×64 RGB images from the Linnaeus 5 dataset. Latency is averaged across ten trials with shaded regions corresponding to standard deviation



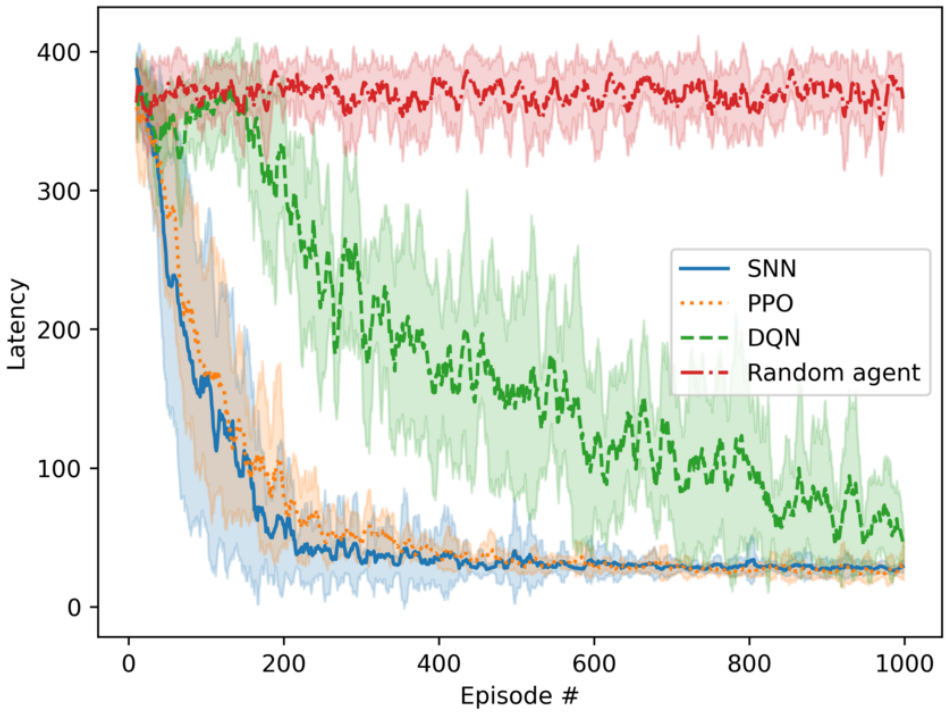
Source: The author, 2023

4.4.4.3 Impact of hyperparameters

This Section provides additional experiments to evaluate the effect of changing individual hyperparameters of the proposed network. For the following experiments, unless otherwise stated, the configuration **a** from Table 7 is used. Note that a solid blue line is always used in the figures for the latency curve of default SNN.

The linear track experiment is used to evaluate a range of parameters of the input and hidden layers. Figure 25a illustrates the influence of the number of neurons in the hidden layer, which is varied from 100 to 2000. While in the main experiments from Section 4.4.4.2 we have used a network with 1000 hidden neurons, this experiment indicates that this parameter could be further optimized without loss of performance. On the other hand, the sparsity parameters n_+ and n_- have significant impact on the network‘s ability to discriminate between different states. This is demonstrated on a linear track experiment in Figure 25b and on the grid-world in Figure 26. On these experiments the amplitude parameters a_+ and a_- do not significantly influence the network‘s latency.

Figure 24 – A comparison of the proposed network with PPO and DQN algorithms on a 20×20 grid-world

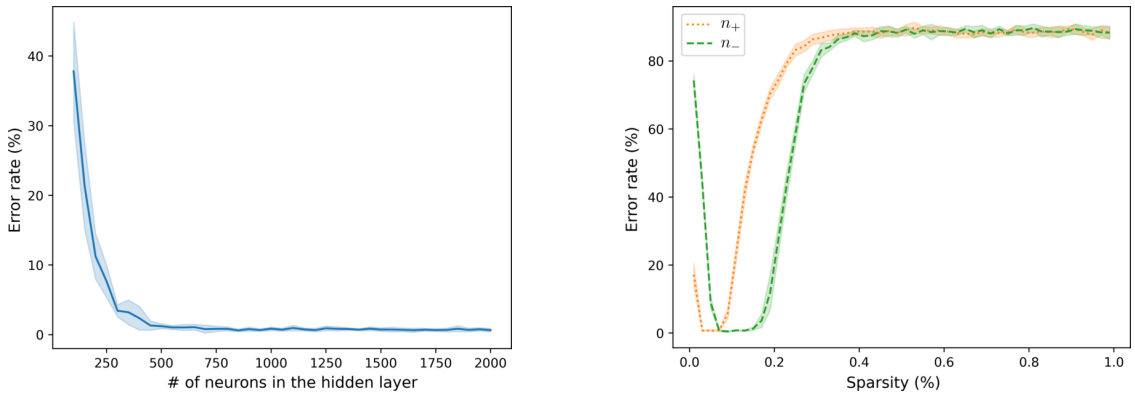


Source: The author, 2023

The number of place cells can also have an impact on the network's ability to learn. This is because the number of neurons in this layers has to be large enough to encode all the states that are necessary to solve a given task. As demonstrated in Figure 27, 50 place cells is enough to solve the 10×10 grid-world task, without noticeable benefit from more neurons in this layer. Note that the proposed approach is currently not well suited for tasks that would require encoding a large set of discrete states. A solution to this limitation is left for future works, as discussed in Section 4.5.1.

The eligibility discharge time τ_e can modulate the learning speed. If τ_e is decreased, the eligibility trace will discharge faster, as described in Equation 3.2. Consequently, this decreases the impact of the states and actions that have occurred before the reward. On the other hand, if the eligibility trace does not discharge quickly enough, the network will commit to suboptimal paths that lead to a rewarded state. This trade-off is illustrated in Figure 28. Since τ_e discharge times are different between hidden-place and place-action traces, we modulate the number of steps that this parameters take to increase to the maximum value.

Figure 25 – The linear track experiment is used to evaluate the impact of the hidden layer size (a) and sparsity of the input-hidden connectivity (b). Shaded region indicates the standard deviation from 10 trials



(a) (b)

Source: The author, 2023

4.4.5 Discussion and summary

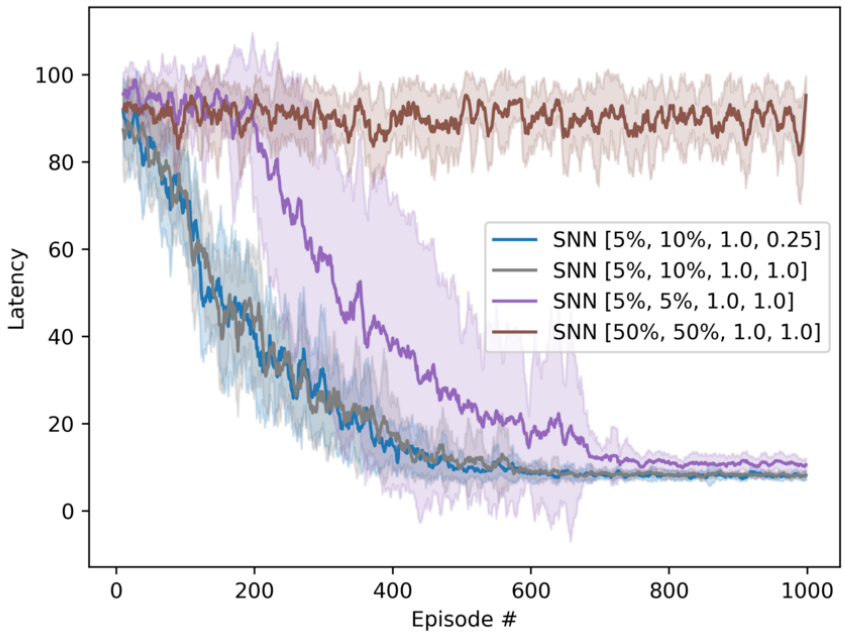
The results presented in the previous section demonstrate the competitiveness of the proposed network, when compared to DRL algorithms. This section contains a summary of the results, as well as a critical comparison to related works.

One aspect of the proposed approach is that the spiking network is able to learn from image observations by using a pre-trained binary CNN (BinaryNet by Courbariaux et al. (2016)). The BinaryNet is trained only once per trial on a small set of images. Moreover, this training set is randomly pulled from a single class of images from the Linnaeus 5 dataset and the observations presented by the grid-world environment are randomly pulled from other four classes in the same dataset. This is akin to the well-known transfer learning approach, where knowledge acquired from one task is used to solve another related problem. A complementary experiment evaluating the scalability of the feature extraction network in terms of input image size if provided in Section A.

The most obvious advantage of separating feature extraction and policy learning is the reduced computational cost, as the convolutional network is used only for inference. The number of plastic synapses is also significantly reduced and is restricted to three layers of the spiking network. It should be noted that a pre-trained feature extraction network is not well suited to deal with a changing environment, as the learned filters may not extract meaningful information. A hybrid approach could be explored to address this limitation, where the feature extraction network is retrained if the spiking agent fails to improve over a long time.

The reduction in the number plastic synapses allows the spiking agent to be trained online, without use of a large memory bank or multiple weight adjustments through gradi-

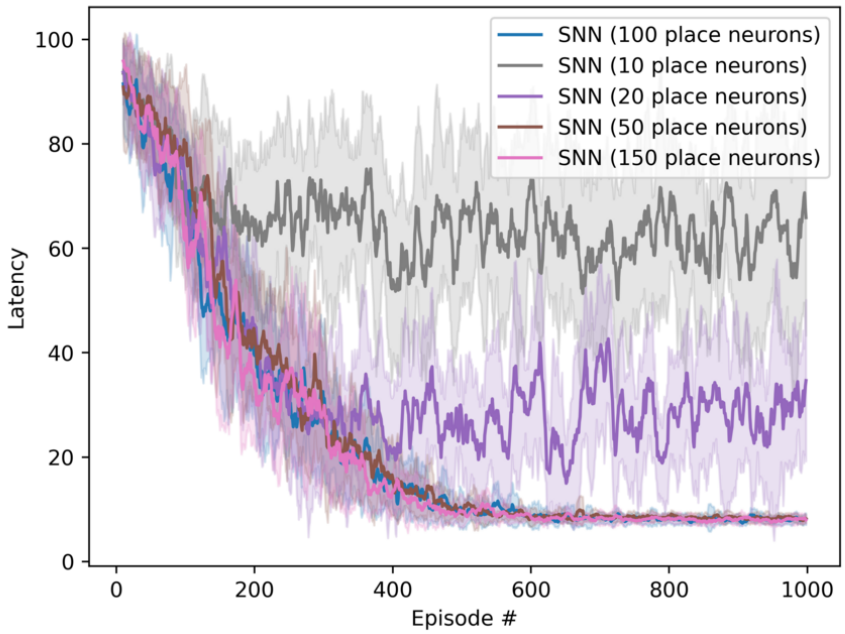
Figure 26 – A qualitative comparison of different input-hidden connectivity schemes



Source: The author, 2023

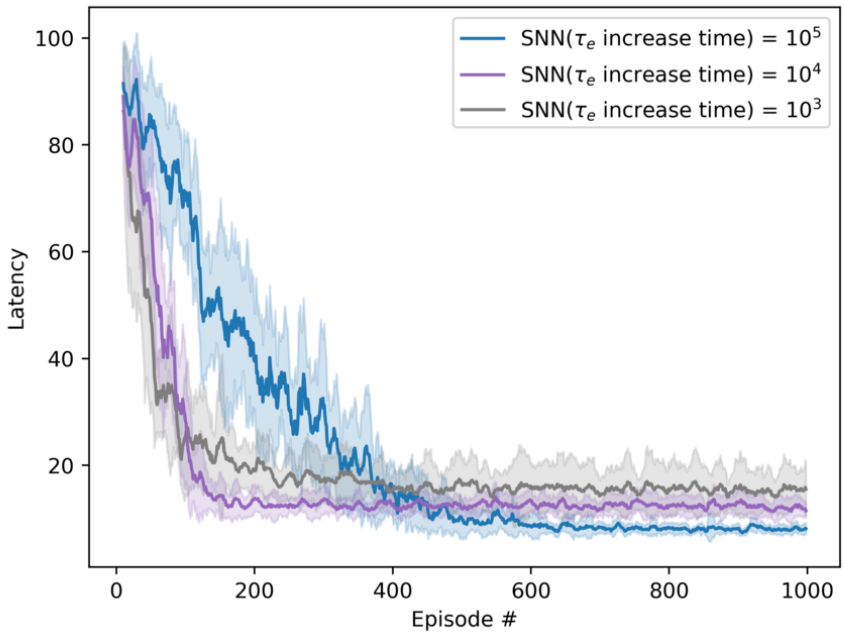
ent descent. It is worth noting that most of related spiking models presented in Section 3.2 are also trained online. However, differently from the actor-critic architectures presented in Frémaux, Sprekeler e Gerstner (2013), Bellec et al. (2020), Chung e Kozma (2020), our network does not perform continuous weight adjustments. Instead, synapses are updated using eligibility traces only upon a discrete reward signal. In the current implementation, this update rule also imposes some limitations: the reward signal is expected to be sparse and the observations are discrete and finite. An actor-critic version of the proposed model will be explored in future works, aiming to achieve rapid learning on a wider range of applications.

Figure 27 – Evaluation of the impact of the number of place neurons on a 10×10 grid-world



Source: The author, 2023

Figure 28 – The parameter "τ_e increase time" can be used to modulate a trade-off between learning speed and final latency



Source: The author, 2023

4.5 CONCLUSION AND FINAL REMARKS

The results presented in this chapter aim to provide a foundation and open prospect for broader applications of spiking networks in reinforcement learning. We combine a binary CNN with a spiking model in order to leverage advantages of both architectures. An automatic hyperparameter optimization strategy is used for the proposed network, as well as the baseline DRL models. After optimization, a single temporal parameter can be adjusted for faster learning, albeit with some stability cost, as illustrated in Figure 28.

The present work advances the field by providing the following contributions:

1. A novel spiking architecture is proposed and demonstrated to learn from sparse and delayed reward with RGB images as inputs.

2. The sparseness of synapses between input and hidden layers of the network is optimized and shown to have a significant impact on learning speed and accuracy.

3. On the evaluated environment, the proposed network is shown to be competitive with state-of-the-art DRL algorithms, both in terms of learning speed and final latency of the learned policy.

4.5.1 Future work

The grid-world environment presented in this work is a versatile toy problem and is shown to present a challenge for the DQN algorithm, assuming a limited time budget. However, environments that are closer to real-world challenges should be explored in future iterations.

One of the main limitations of the proposed network is that the place layer requires a large number of neurons to encode discrete states in the environment. Future works should focus on a better hidden-place plasticity model to allow the same place neuron to encode similar states. Some other possible directions for future endeavors are presented below:

- Improved exploration – a more sophisticated approach to exploration could allow to learn policies from very sparse rewards. Traditional RL algorithms are improved by hindsight experience replay (ANDRYCHOWICZ et al., 2017), which could be a source of inspiration for a novel spiking architecture.

- Learning of hidden representations – while the current work makes use of a hidden layer and STDP for state representation, this approach should be expanded to continuous observations.

- Automatic regulation of temporal parameters – the synaptic discharge time and other temporal parameters can significantly influence the performance of the network. Instead of the offline optimization strategy presented in this chapter, these

parameters could be adjusted online by a heuristic approach. For instance, the synaptic discharge rate could be regulated by the rate at which the agent receives a reward. Thus, a given policy could quickly revert to a more random search if the environment changes or the policy fails in other ways.

5 A NEUROMORPHIC ARCHITECTURE FOR REINFORCEMENT LEARNING FROM REAL-VALUED OBSERVATIONS

This chapter was originally made available as pre-print on arXiv (CHEVTCHENKO et al., 2023).

5.1 INTRODUCTION

In our previous works (CHEVTCHENKO; LUDERMIR, 2020; CHEVTCHENKO; LUDERMIR, 2021), we have demonstrated that an SNN trained through STDP can compete with state-of-the-art RL algorithms to solve classical control and RL tasks involving distal rewards and image observations. Additionally, we show that sparse connectivity becomes increasingly important with larger state spaces. The proposed approach differs from recent related ones (BELLEC et al., 2020; TANG; KUMAR; MICHMIZOS, 2020; CHUNG; KOZMA, 2020; KUMAR et al., 2021), as we use the STDP plasticity for two distinct tasks in the network: i) state space reduction through hidden and place layers and ii) policy learning in the last layer. Furthermore, the network architecture used in the present work does not require a rate-coded approach, reducing the latency and learning speed when compared to related models.

In this chapter, we introduce a novel neuromorphic architecture designed to tackle reinforcement learning problems with real-valued observations. Drawing inspiration from prior works (AFSHAR et al., 2020; BETHI et al., 2022), our network incorporates clustering layers and introduces modulation by a global signal through eligibility traces. We assess the effectiveness of the proposed model against a tabular actor-critic algorithm with eligibility traces and a state-of-the-art DRL model, Proximal Policy Optimization (PPO). The results demonstrate that our network provides an appealing trade-off in terms of computational efficiency and hardware implementation requirements when compared to PPO on three classic RL control tasks.

This chapter is structured as follows. A review of related literature and comparison to the present work is provided in Section 5.2. The proposed model is described in Section 5.3 and is experimentally compared to baseline models in Section 5.4. Concluding remarks and discussion of possible future endeavors can be found in Section 5.5. Additional experiments and a demo code are provided as supplementary materials.

5.2 RELATED WORKS

Deep learning has shown impressive results in the domain of Reinforcement learning and has surpassed all other conventional methods to be the state-of-the-art architectures for training autonomous agents on sparse reward structures. However, the training procedures

for Artificial Neural Networks (ANNs) have always been expensive in terms of computation and memory consumption. Deploying ANNs at the edge with low-power constraints and training them online still remains a challenge due to the same reason. Recent advances in neuromorphic computing and training Spiking Neural Network (SNN) architectures offer novel solutions to building low-power machine intelligence. SNN architectures that use binary valued spikes (or events) for both computation and communication provide excellent energy efficiency benefits. With the uptake in the adoption of neuromorphic sensors like DVS (LICHTSTEINER; POSCH; DELBRUCK, 2008) and ATIS (POSCH; MATOLIN; WOHLGENANNT, 2010), event-driven machine learning algorithms are also being explored to explore low-latency and low-power applications of neuromorphic computing (GALLEGO et al., 2020).

Training SNN architectures is still an active area of research, and many attempts have been made to approximate the error backpropagation and gradient descent techniques used to train ANNs and train SNNs. For example, Bellec et al. (2020) proposed a spiking neural network learning rule called e-prop that can be applied to recurrent spiking networks. This rule is shown to approximate the performance of an LSTM network trained via backpropagation through time on two discrete Atari games. Other methods use surrogate gradients that use stand-in differentiable alternatives for non-differentiable parts of SNN architectures (NEFTCI; MOSTAFA; ZENKE, 2019). For instance, Akl et al. (2023) propose a combination of DRL frameworks with a spiking architecture through the use of surrogate gradients and the backpropagation through time (BPTT) algorithm. However, these methods also face the same problem of requiring energy-intensive computational resources. The error back-propagation approximations are also not bio-plausible as they would require symmetric backward pathways that transfer precise continuous-valued gradients and, in some cases, non-causal operations like Back-Propagation Through Time (BPTT). Bio-plausible local learning rules that can train individual nodes in SNN architectures using the only information available at each neuron can offer potential solutions for exploiting in-memory computation in neuromorphic hardware. Spike-Timing-Dependent Plasticity (STDP) has been the most commonly used local learning rule for SNN architectures to perform unsupervised learning. STDP can train spiking neurons to learn the spatio-temporal features that reflect the statistics of the input spiking data. The Hebbian learning rule has primarily been used in unsupervised learning settings to learn some useful features for tasks like pattern classification (DIEHL; COOK, 2015). However, unsupervised learning rules like STDP neglect any information related to the "success", "failure", or "novelty" of the inputs and outputs (FRÉMAUX; GERSTNER, 2016).

R-STDP (Reward-modulated Spike Timing-Dependent Plasticity) is a Spiking Neural Network (SNN) learning rule that governs the changes in the strength of connections between neurons such that the plasticity of synapses is modulated by reward signals that reinforce or weaken the connections between neurons based on the timing of their spikes.

Izhikevich (2007), Florian (2007) and Legenstein, Pecevski e Maass (2008) independently laid the groundwork for R-STDP modulated spiking networks. In order to demonstrate the computational and temporal capabilities of this approach, a fully connected multilayered network of spiking neurons is shown by Florian (2007) to solve a temporally coded XOR problem with a delayed reward. Vasilaki et al. (2009) used a reward-modulated STDP learning rule on action layer in combination with an input place cell layer to perform reinforcement learning on a Morris water maze puzzle. While theoretical models have used reward-modulated STDP for over a decade, recent research has provided experimental evidence of the role of eligibility traces in Reinforcement Learning (RL)(GERSTNER et al., 2018). The R-STDP rule used in the present work is a simplified version of RMSTDPET found in the work by Florian (2007). Potjans, Diesmann e Morrison (2011), and later Frémaux, Sprekeler e Gerstner (2013) propose actor-critic models with temporal difference (TD). A more general three-factor learning rule is later introduced by Frémaux e Gerstner (2016).

A key component of reinforcement learning is to develop useful internal representations of complex environments to evaluate and utilize the current state of the environment to decide on actions that provide maximum future rewards (SUTTON; BARTO, 2018). Various internal representations for inputs have been used by the SNN solutions for reinforcement learning. One strategy is to use manually selected partitions of the input space to generate spiking inputs for various states. Potjans, Morrison e Diesmann (2009) and Jitsev, Morrison e Tittgemeyer (2012) used a fixed cluster of neurons that represent the state space and individually fire for each particular state. Frémaux, Sprekeler e Gerstner (2013) used pre-determined place cells to act as input to their continuous time spiking actor-critic architecture. Friedrich, Urbanczik e Senn (2014) used population coding for the input layer representation. These methods use a manually chosen method of partitioning the input space or use a population that uniformly covers the entire input space of any given environment. A drawback of these early models is that the implemented networks fully encode the observed state in the input layer, functionally similar to classical tabular RL algorithms. In other words, each neuron in the input layer is used to encode a specific state of the environment. Also, this approach does not scale well with the dimensions of the input space. This limitation in scalability is addressed in our previous paper (CHEVTCHENKO; LUDERMIR, 2020) by a four-layered network with a hidden layer inspired by place cells.

Other approaches to represent the input space include using the reservoir computing paradigm (Schrauwen, Verstraeten e Campenhout (2007);Lukoševičius e Jaeger (2009)). Reservoir computing involves projecting the low dimensional inputs to a high dimensional representational space by using a population of recurrently connected neurons, such that states not linearly separable in the input space can be separated in the representational space (MAASS; NATSCHLÄGER; MARKRAM, 2002). Weidel, Duarte e Morrison (2021) used reservoir computing with unsupervised plasticity on the inputs and apply reward modu-

lated learning on the output layer that generated actions. However, most of these SNN architectures predominantly use rate coding, which does not offer the same level of energy efficiency that sparse temporal coding provides.

Another promising avenue for potential low-power spike-based learning algorithms is event-driven neuromorphic algorithms. Neuromorphic algorithms and hardware are primarily targeted towards scalability through collocated processing and memory, and low-latency computation (SCHUMAN et al., 2022). Neuromorphic event-driven algorithms emulate spiking neuron architectures that are tailored for computational efficiency and performance. HFirst algorithm (ORCHARD et al., 2015) is an example of a multi-layered network architecture that is based on the HMAX algorithm (SERRE et al., 2007) which itself inspired by visual cortex to learn features from event-based data using an event-driven approach of processing data. The Hierarchy of Event-based Time Surfaces (HOTS) (LAGORCE et al., 2016) is another unsupervised multi-layered feature extraction algorithm that uses the same neuron update rule based on the cosine distance of weights to the input representation introduced in Ballard e Jehee (2012) to model sensory features in cortical neurons. The HOTS algorithm uses multiple layers to extract spatio-temporal patterns at multiple hierarchical levels with different time constants by performing clustering. Afshar et al. (2020) introduced an unsupervised feature extraction algorithm called Feature Extraction using Adaptive Selection Thresholds (FEAST), which is also capable of learning hierarchical spatio-temporal features by using adaptive thresholds for each neuron to promote equal activation of neurons.

The majority of the neuromorphic feature extraction algorithms are primarily targeted toward performing tasks on event-based data. The FEAST method has been used for a range of applications such as event-based object tracking Ralph et al. (2022), activity-driven adaptation in SNNs Haessig et al. (2020), and spoken digits recognition task Xu et al. (2022). Recently a generalization of the FEAST algorithm has been proposed in Bethi et al. (2022) to perform supervised learning on spiking data using reward and punishment of neurons through threshold adaptation. We can utilize the same neuromorphic principles and apply these architectures to the reinforcement learning domain to partition the input state space that can dynamically adapt to the reward structure and performance of the agent. In this work, we modify the neuron layers proposed in the Bethi et al. (2022) to simultaneously perform unsupervised clustering of the input space and modulate it based on the ongoing TD-error of an actor-critic agent that utilizes the learned representation. We show that partitioning achieved by the neurons represents different parts of the input space with varying degrees of density and resolution depending on the ongoing performance of the actor-critic agent using the representation. We demonstrate the quality of the input space representations by using a conventional tabular actor-critic algorithm and applying it to various environments.

5.3 THE PROPOSED ARCHITECTURE

The proposed network has two main components: i) an initial clustering layer(s) for dimensionality reduction and discretization of the input signal and ii) actor-critic neurons for learning from the previously reduced and discretized state space representation. The following sections provide a detailed description of each layer and corresponding plasticity rules.

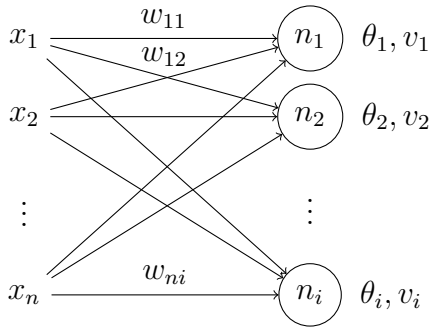
5.3.1 Input layer

The input signal is represented by a vector of real-valued observations $\overrightarrow{x} = [x_1, x_2, ..., x_n]$. These are provided by the environment in discrete time-steps at time t, along with the reward signal $r(t)$. Considering N_i input neurons, each neuron can be fully connected to the input signal through an array of synaptic weights, as illustrated in Figure 29. Euclidean distance is used as a distance metric to find the closest neuron i with weights $\overrightarrow{w_i}$ matching the input context $\overrightarrow{x}$. Each neuron i also contains a scalar threshold parameter θ_i that is used for the selection of the closest neuron. The value v_i of a neuron in the input layer for an input vector is calculated as the Euclidean distance between input and weight vectors:

$$v_i(t) = ||\overrightarrow{w_i}(t) - \overrightarrow{x}(t)||, \tag{5.1}$$

where $v_i(t)$ is the value of the neuron i at time t, $\overrightarrow{w_i}$ are the weights connecting the neuron i to the input vector $\overrightarrow{x}$. The value v_i of every neuron is compared to its threshold θ_i to check if the neuron is eligible to be activated. A winner-take-all (WTA) activation is adopted for an entire layer or a group of neurons within the same layer. Thus, only the neuron with the least euclidean distance within the eligible neurons will produce a spike for the next layer. The winning neuron should have the least value among the neurons with values within their corresponding thresholds.

Figure 29 – Illustration of the input layer of the proposed model

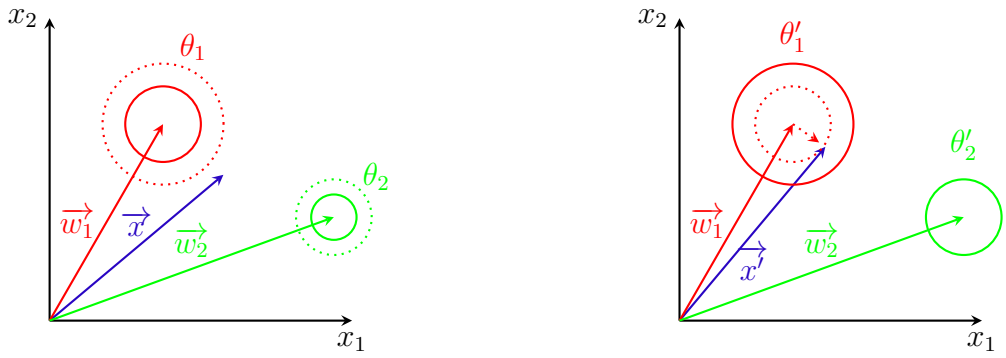


Source: The author, 2023

The weight modulation used in this layer has two drivers: i) adaptation to the input signal and ii) modulation by the feedback from the actor-critic layer. The first adaptation rule is based on Feature Extraction with Adaptive Selection Thresholds (FEAST), proposed by Afshar et al. (2020). To this end, each neuron's threshold parameter θ_i is adaptable.

Consider a two-dimensional input vector $\overrightarrow{x} = [x_1, x_2]$, as illustrated in Figure 30: (a) The input vector $\overrightarrow{x}$ is outside the threshold regions of both neurons. The neurons become more sensitive by increasing the thresholds. In Figure 30 (b), a new input vector activates neuron 1. This results in adjustments of both weight and threshold of the active neuron. The input signal $\overrightarrow{x}$ and the weights of two input neurons are represented by two-dimensional vectors. The threshold of each neuron is a scalar and represents the receptive field of a neuron in the shape of a circle around the weight vector.

Figure 30 – An illustration of synaptic plasticity rule based on FEAST (AFSHAR et al., 2020) using Euclidean distance. A two-dimensional input is considered. (a) The input vector $\overrightarrow{x}$ is outside the threshold regions of both neurons. The neurons become more sensitive by increasing the thresholds. (b) A new input vector activates neuron 1. This results in adjustments of both the weights and the threshold of the active neuron.



Source: The author, 2023

An initial input signal $\overrightarrow{x}$, depicted in Figure 30a, falls outside the activation region of both neurons (solid circles). Thus, the value obtained by Equation 5.1 is higher than θ_1 and θ_2. When this occurs, all neurons in the layer have their thresholds increased by the parameter θ_o. This increases the sensitivity of the layer to any input signal, making the activation within the threshold more likely in the future, as indicated by the dotted circles.

On the other hand, Figure 30b illustrates a new input vector $\overrightarrow{x'}$ that activates the neuron 1. This is because the new value, i.e. the distance between input and weight vectors, is within the threshold of the neuron. In order to increase the probability of

activation of the same neuron in the future, when presented with similar inputs, both threshold and weight are adjusted. The threshold is decreased and the weight vector moves in the direction of $\overrightarrow{x}$, illustrated by the dotted circle and arrow in Figure 30b. This makes the neuron more specialized and less likely to be activated by more distant inputs. The above adaptation, applied to activated neurons, is described by:

$$\Delta\theta_i = v_i(t) - \theta_i(t), \tag{5.2}$$

$$\Delta\overrightarrow{w_i} = \overrightarrow{x}(t) - \overrightarrow{w_i}(t), \tag{5.3}$$

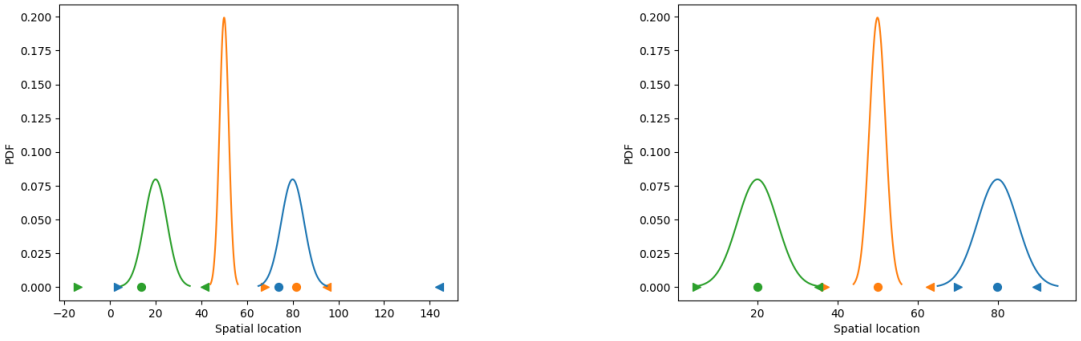
$$\theta_i(t+1) = \theta_i(t) + \eta_{th} * \Delta\theta_i, \tag{5.4}$$

$$\overrightarrow{w_i}(t+1) = \overrightarrow{w_i}(t) + \eta * \Delta\overrightarrow{w_i}, \tag{5.5}$$

where $\Delta\theta_i$ and $\Delta\overrightarrow{w_i}$ are update values for the threshold and weights of neuron i and the respective update rates for each are η_{th} and η.

Figure 31 illustrates this adaptation process on a simulated dataset with three probability density functions (PDF). Three neurons are initialized with random weights and thresholds, as illustrated in Figure 31a. The weight of each neuron is depicted as a colored dot and the threshold region is delimited by same-colored arrows. The final configuration is depicted in Figure 31b, after 10^4 observations and using both η_{th} and η equal to 10^{-2}.

Figure 31 – Clustering of neurons in the input layer on a simulated dataset with three Gaussian distributions and three input neurons. The weights of the three neurons are depicted by the colored dots and the corresponding thresholds are represented by the same-colored arrows



(a) Initial position

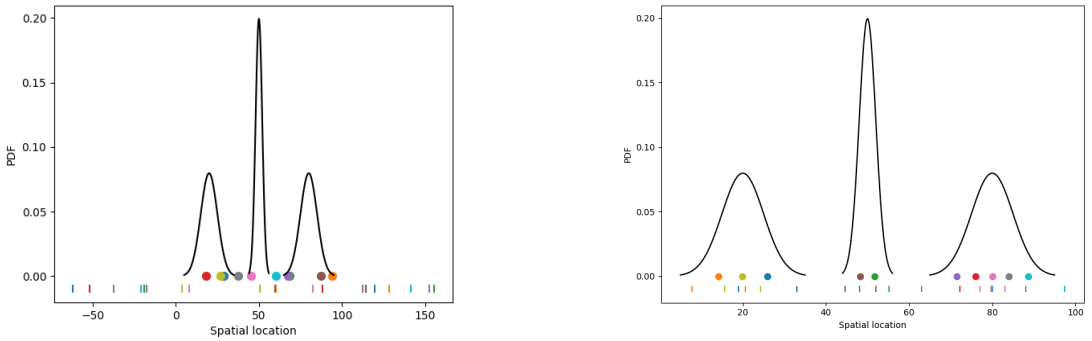
(b) After adaptation

Source: The author, 2023

Note that if more neurons are added to the clustering layer, the adaptation process described by Equations 5.2 to 5.5 will result in lower threshold values and a more granular representation of the observed state space. This is experimentally demonstrated in Figures

32a and 32b by using the same set of parameters as in the previous experiments, but increasing the number of neurons from 3 to 10.

Figure 32 – Clustering of neurons in the input layer on a simulated dataset with three Gaussian distributions and ten input neurons. The positions of the colored dots on the X-axis represent the weights of the ten neurons used in the experiment. For better legibility, thresholds are represented as vertical lines of the same color below the dots



(a) Initial position

(b) After adaptation

Source: The author, 2023

In addition to the weight and threshold modulation above, the proposed model implements modulation by feedback from the actor-critic layer. This is similar to the change in Equations 5.4 and 5.5, but η_{th} and η are replaced by temporal difference error δ and neural trace c_i. This modulation is described in more detail in Section 5.3.2.

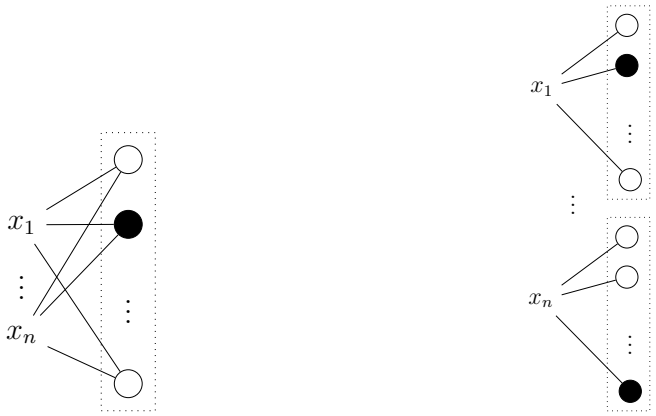
The input layer neurons can encode the real-valued vector $\overrightarrow{x}$ in a number of ways. In the present work we consider two possible configurations, as depicted in Figure 33.

Note that the configuration in Figure 33b produces multiple active neurons at each time-step, corresponding to the size of the input vector $\overrightarrow{x} = [x_1, x_2, ..., x_n]$. In this case, a hidden clustering layer is used to provide a single winner neuron for the actor-critic layer. The WTA activation allows for a single neuron from this second layer to produce a spike at each time. The synapses between the first and second clustering layers are subject to the same modulation, as described by Equations 5.2 to 5.5.

5.3.2 Actor-critic layer

The actor-critic layer is a neuromorphic implementation of the tabular TD(λ) algorithm (SUTTON, 1988). The actor is composed of N_a neurons, each representing a single discrete action. When a spike is produced by a single presynaptic neuron j, the resulting potential of an action neuron i is equal to the synaptic value w_{ij}. The action is then selected based on the highest potential, with the addition of random exploration:

Figure 33 – Diagrams of evaluated configurations for encoding the input vector by the first layer. (a) The input is fully connected to the neurons in the first layer with WTA activation. (b) A group of neurons if fully connected to a single input scalar and WTA is applied to each group separately. Grouped neurons within a dotted rectangle indicate winner-take-all (WTA) activation where the neuron with the highest potential propagates a pulse to the next layer



Source: The author, 2023

$$a(t) = \begin{cases} \arg\max_i w_{i,j}, & \text{with probability } 1 - \epsilon \\ random, & \text{with probability } \epsilon, \end{cases} \tag{5.6}$$

where ϵ is the exploration probability. At the beginning of an episode ϵ is initialized at a maximum value 1 and is then gradually decreased to a small final value ϵ_{min}. The rate of decrease and the final baseline value are hyperparameters used to balance exploration and exploitation.

In the proposed system an action can be the result of a single or multiple neurons in the action layer spiking by using WTA activation. For instance, a movement in two dimensions can be encoded by two independent groups of neurons, each representing a one-dimensional action.

The value of a hidden state is represented by the weight connecting the value neuron to the hidden layer. Two separate synaptic eligibility traces are implemented for the actor and critic connections. The update is described by Equations 5.7 and 5.8:

$$\overrightarrow{c_c}(t+1) = \overrightarrow{c_c}(t) - \frac{\overrightarrow{c_c}(t)}{\tau_c}, \tag{5.7}$$

$$\overrightarrow{c_{a\,i}}(t+1) = \overrightarrow{c_{a\,i}}(t) - \frac{\overrightarrow{c_{a\,i}}(t)}{\tau_a}, \tag{5.8}$$

where τ_a and τ_c are time constants for the actor and critic traces respectively. Correspondingly, actor and critic trace vectors are denoted as $\overrightarrow{c_{a\,i}}$ and $\overrightarrow{c_c}$. Note that $\overrightarrow{c_{a\,i}}$ relates to the action neuron i and c_{aij} is set to one when an action neuron i and hidden neuron j fire at the same discrete time step.

After an action is selected, the environment provides the next observation $\overrightarrow{x}(t+1)$ and reward signal $r(t+1)$. Based on this feedback, the temporal difference (TD) error is calculated as follows (SUTTON; BARTO, 2018):

$$\delta = r(t+1) + \gamma V(t+1) - V(t), \tag{5.9}$$

where $V(t)$ is the value of the hidden state at time t and γ is a discount value. Based on the TD error, value and action weights are updated:

$$\overrightarrow{w_v}(t+1) = \overrightarrow{w_v}(t) + \overrightarrow{c_c}(t) * \eta_c * \delta, \tag{5.10}$$

$$\overrightarrow{w_{ai}}(t+1) = \overrightarrow{w_{ai}}(t) + \overrightarrow{c_{a\,i}}(t) * \eta_a * \delta, \tag{5.11}$$

where $\overrightarrow{w_v}$ is a vector of values, $\overrightarrow{w_{ai}}$ are weights arriving at the action neuron i. Constants η_c and η_a are update rates for the critic and actor respectively. Finally, weights and thresholds of clustering layers are also modulated by the TD error:

$$\theta_i(t+1) = \theta_i(t) + \eta_{td} * \Delta\theta_i * |\delta| * c_i(t), \tag{5.12}$$

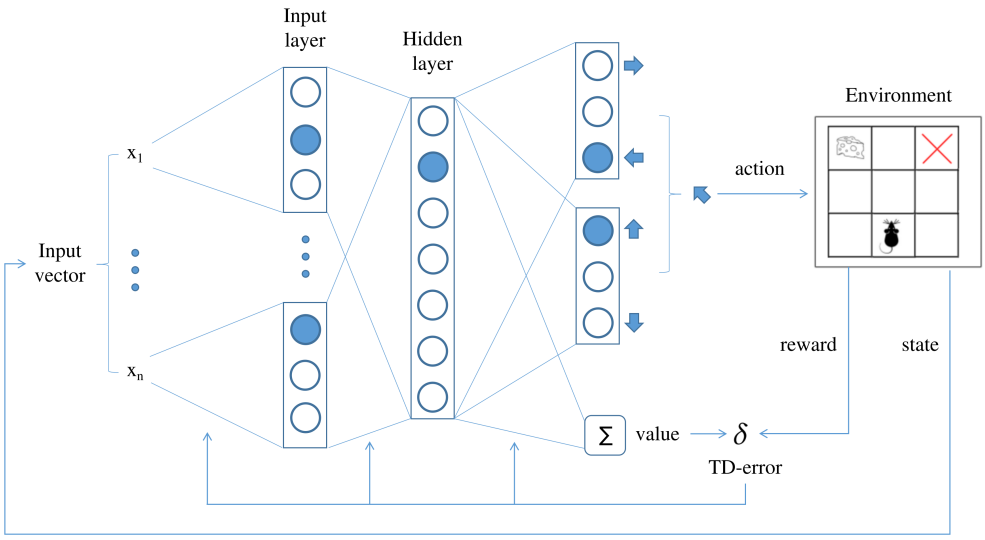
$$\overrightarrow{w_i}(t+1) = \overrightarrow{w_i}(t) + \eta_{td} * \Delta\overrightarrow{w_i} * |\delta| * c_i(1), \tag{5.13}$$

where η_{td} is the update rate and $c_i(t)$ is the activation trace of the postsynaptic neuron i. Following Equation 5.8, this trace is set to one each time neuron i produces a spike and decays exponentially with time constant τ_i. This update occurs concurrently with unsupervised clustering described by Equations 5.4 and 5.5.

With the above update rule lower absolute TD error causes the neuron to follow the unsupervised clustering rule from Equations 5.4 and 5.5 and illustrated in Figure 30. Conversely, higher positive or negative TD errors make the model more likely to retain the current feature representation.

An overview of the proposed network is illustrated in Figure 34. In this example, the first clustering layer is divided in groups of tree neurons, each with a WTA activation. Each of these groups is connected to a pair of input values and thus can be viewed as two-dimensional clustering. The activation trace from this first layer is usually a higher dimensional vector than the input which is then converted to a one-hot vector by the hidden layer. Both the first and second layers implement the FEAST clustering rule proposed by Afshar et al. (2020), as well as additional modulation by TD error, described by Equations 5.12 and 5.13.

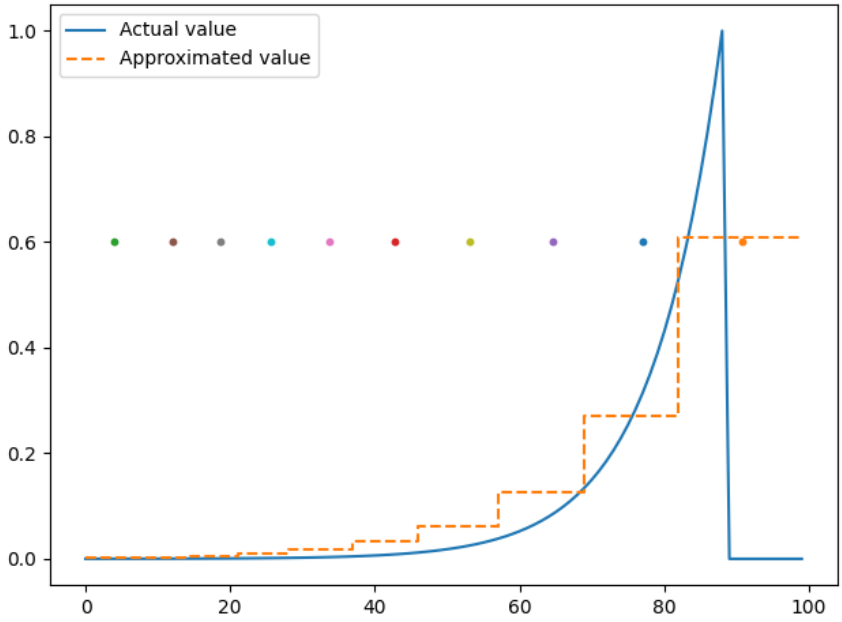
Figure 34 – An overview of the presented network. In this example, each input scalar x_i is connected to a group of three neurons in the first clustering layer. A WTA activation is applied independently on each group and a resulting multi-hot vector is processed further by the hidden clustering layer. Another WTA activation function, this time applied to the entire layer, results in a single active neuron which represents a distinct hidden state. Orthogonal actions are independently encoded by subsequent groups of neurons and the value corresponding to the hidden state is computed based on sum of the synaptic weights. Eligibility traces are attached to both neurons and synapses for weight and threshold modulation



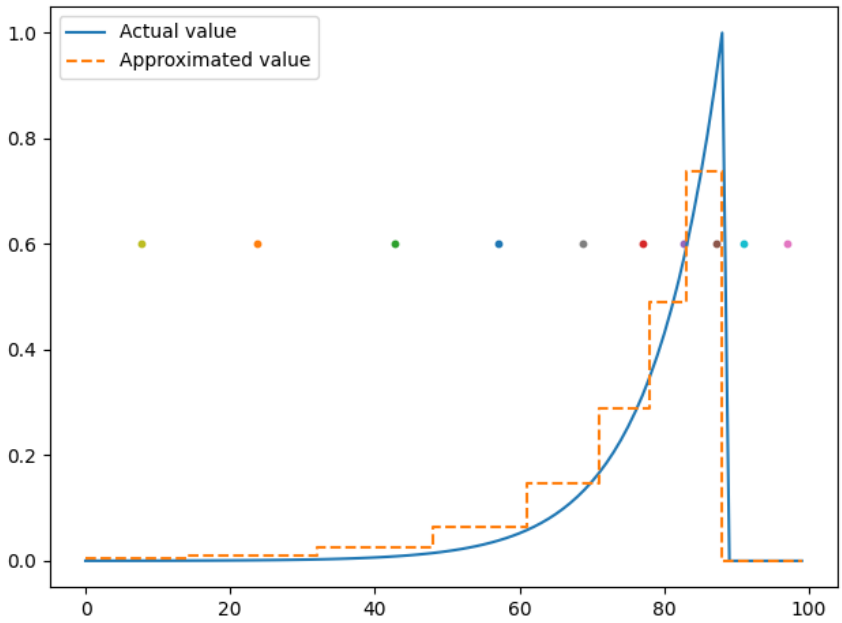
Source: The author, 2023

A linear track experiment is used to illustrate the role of TD-modulated clustering. In this environment, the agent travels through a fixed track from position 0 to 100 and each time-step moves the agent to the right by one unit. A single reward of +1 is provided at position 90 and once the end of the track is reached, the environment restarts at 0. Considering a discount $\gamma = 0.9$, the value corresponding to each position is represented by a solid blue line in Figure 35. Since in the proposed architecture, the state values are estimated from a hidden layer, a discrete approximation of a real valued curve is formed by the ten hidden neurons. This approximation is represented by a dotted orange line and the solid dots correspond to the weights of these hidden neurons after 500 episodes. Note that if no TD-modulation is used, the weights would converge to a uniform representation of the observed states, as in Figure 35a. On the other hand, by adding modulation through TD-error, the weights will be skewed towards the states that produce the highest TD-error, which is illustrated by a more accurate approximation of the value curve in Figure 35b.

Figure 35 – An illustration of the role of TD-modulated clustering for a more accurate value representation. Position of the colored dots on the X-axis represent the weights of the ten neurons used in the experiment. Thresholds are omitted for legibility



(a)



(b)

Source: The author, 2023

5.4 EXPERIMENTAL EVALUATION

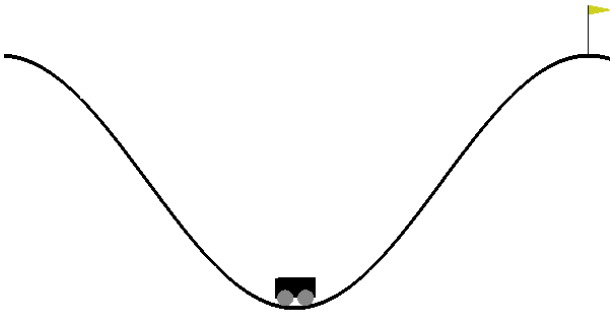
The proposed network is experimentally compared to a classical tabular actor-critic algorithm with eligibility traces (TAC), as well as a state-of-the-art deep reinforcement learning (DRL) algorithm PPO. Three simulated RL environments from OpenAI Gym (BROCKMAN et al., 2016) are used for this evaluation: mountain car, Cart-pole and acrobot. The sections below provide a more detailed description of these environments and baseline algorithms.

5.4.1 RL environments

5.4.1.1 Mountain car

The Mountain car is a classic RL control problem, illustrated in Figure 36. The environment consists of a car that is stuck in a valley between two hills. The car has limited power and is unable to climb the hills directly. The goal of the agent is to learn how to control the car so that it can reach the top of the hill on the right.

Figure 36 – The mountain car environment



Source: Adapted from Brockman et al. (2016)

This environment has a two-dimensional state space, consisting of the position and velocity of the car. The position is a continuous value that ranges from -1.2 to 0.6, while the velocity ranges from -0.07 to 0.07. The action space is discrete, consisting of three possible actions: push the car to the left, push the car to the right, or do nothing.

The reward function in the Mountain car environment is designed to encourage the agent to reach the top of the hill on the right. The agent receives a reward of -1 at each time step until it reaches the goal. This penalization is added to encourage the agent to reach the goal as quickly as possible. Once the agent reaches the goal, the episode ends and the agent receives a reward of 0.

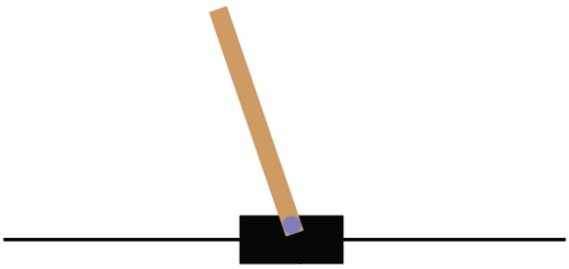
This is a non-trivial problem because of the long-term dependencies involved. The agent must learn to build up momentum by moving back and forth in the valley before it

can climb the hill on the right. This requires the agent to balance short-term rewards with long-term goals. Importantly, the goal state is highly unlikely to be reached by chance, i.e. random exploration, within the time limit of 200 steps.

5.4.1.2 Cart-pole

The Cart-pole environment in OpenAI Gym (BROCKMAN et al., 2016) is another classic RL problem, illustrated in Figure 37. The environment consists of a cart and a pole that is attached to the cart by a free-moving joint. The goal of the agent is to balance the pole on top of the cart for as long as possible.

Figure 37 – The Cart-pole environment



Source: Adapted from Brockman et al. (2016)

The state space of the Cart-pole environment is four-dimensional, consisting of the position and velocity of the cart, and the angle and angular velocity of the pole. The cart position and pole angle are continuous values, as well as the respective velocities. The action space is discrete, consisting of two possible actions: move the cart to the left or move the cart to the right.

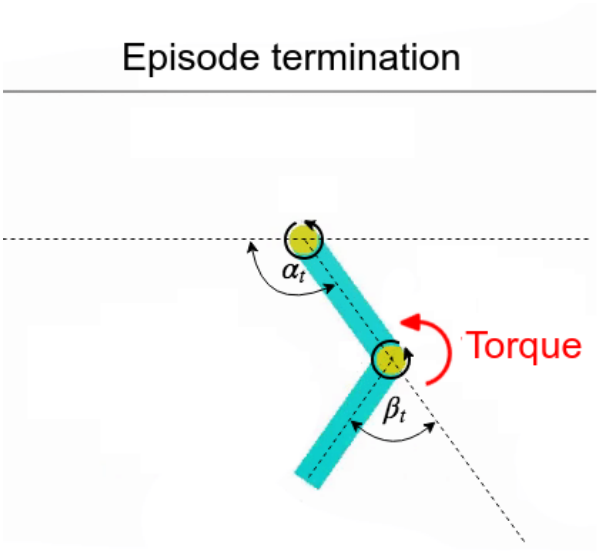
The reward function in the Cart-pole environment is designed to encourage the agent to balance the pole on top of the cart for as long as possible, limited to 500 steps. The agent receives a reward of +1 at each time step while the pole remains upright. The episode ends when the pole falls over or when the cart moves too far to the left or right. Once the episode ends, the agent receives a reward of 0.

The Cart-pole is a challenging RL problem because of its instability. The agent must learn to balance the pole while also avoiding moving the cart too far to either side.

5.4.1.3 Acrobot

The setup is illustrated in Figure 38 and the goal is to lift the tip of the robot to a certain level. The environment provides sine and cosine of both joint angles, as well as the respective angular velocities. The second joint is weakly actuated and the system includes gravitational pull. To solve this task, the agent has to consistently swing the actuated joint, building up the energy.

Figure 38 – The acrobot environment



Source: Adapted from Brockman et al. (2016)

The reward function in the Acrobot environment is designed to encourage the agent to swing the pendulum up to a vertical position. The agent receives a reward of -1 at each time step, as with the other environments, this encourages the agent to reach the goal as quickly as possible. When pendulum reaches the goal position the episode ends and the environment provides a reward of 0.

5.4.2 Baseline algorithms

5.4.2.1 TAC

The tabular actor-critic algorithm is based on a classical TD(λ) reinforcement learning algorithm used to estimate the optimal value function of an MDP, given a set of states, actions, and rewards (SUTTON, 1988). It is a variant of the TD-learning algorithm that incorporates eligibility traces, which allow for the accumulation of TD-errors over multiple time steps.

At each time step, the algorithm updates the value function estimate of the current state s using the TD-error, defined in Equation 5.9. This update is weighted by the eligibility trace, which is a measure of the importance of the state s in the current episode.

The eligibility trace is updated using the decay parameter λ and the discount factor γ at each time step, so that states that occur closer to the current time step are given more weight in the update. This allows the algorithm to incorporate information from previous time steps into the current update and improve the accuracy of the value function estimate.

The algorithm repeats this process for a fixed number of episodes, updating the value

function estimate and eligibility traces at each time step. The epsilon-greedy exploration strategy is used to balance exploration and exploitation.

5.4.2.2 PPO

Proposed by Schulman et al. (2017), Proximal Policy Optimization (PPO) is a RL algorithm used for training policy-based models, which directly optimize the policy function that maps states to actions. It is a variant of the policy gradient methods, which aim to maximize the expected reward of an agent by adjusting the parameters of the policy function.

PPO uses a surrogate objective function that constrains the change in the policy parameters to be small, so that the agent does not deviate too far from the previous policy. This makes the training process less brittle and prevent the agent from making too large policy changes. The PPO algorithm uses a clipping technique to enforce this constraint on the policy updates. At each iteration, the algorithm computes the ratio between the new and old policy probabilities for the observed state-action pairs, and uses this ratio to compute a surrogate objective function. The surrogate objective function is then optimized using a gradient descent algorithm, subject to a constraint that limits the size of the policy update.

The size of the policy update is controlled by a hyperparameter ϵ, which represents the maximum amount that the policy parameters can change in a single update. If the ratio of the new and old policy probabilities exceeds $1 + \epsilon$, then the update is clipped to a maximum value of $1 + \epsilon$. Similarly, if the ratio is less than $1 - \epsilon$, then the update is clipped to a minimum value of $1 - \epsilon$. This helps to prevent the policy from changing too much and ensures that the updates are consistent with the previous policy.

Analogously to the actor-critic model in the proposed network, PPO also uses a value function to estimate the expected reward of each state.

5.4.3 Hyperparameters

This section presents the main hyperparameters that were found to have an impact on each algorithm's performance during exploratory trials. A widely used optimization framework *Optuna* (AKIBA et al., 2019) is employed to iteratively search for a good set of hyperparameters for the proposed network and both of the baseline algorithms. More specifically, the Tree of Parzen estimators (TPE) (BERGSTRA et al., 2011) algorithm is used for sequential optimization. After an initial random exploration for 10 trial, TPE aims at choosing a set of parameters that maximize an objective function. If a model with the same set of parameters is evaluated more than once, the final score is an average of all the trials for this set.

Table 8 – Hyperparameter search space for the TAC baseline

Hyperparameter	Search space
# of bins per dimension	[5, 10(**b**,**c**), 20(**a**)]
ϵ_{min}	[0.01(**a**), 0.05(**b**,**c**), 0.1]
ϵ decay time (# of episodes)	[100, 200(**b**,**c**), 500(**a**)]
Discount factor	[0.9(**c**), 0.95(**a**,**b**), 0.99]
Actor learning rate	[10^{-3}, 10^{-2}(**b**,**c**), 10^{-1}(**a**)]
Critic learning rate	[10^{-3}, 10^{-2}(**b**,**c**), 10^{-1}(**a**)]
τ_a	[1(**a**,**b**,**c**), 10, 20]
τ_c	[1, 10, 20(**a**,**b**,**c**)]

Source: The author, 2023

Table 9 – Hyperparameter search space for the PPO baseline

Hyperparameter	Search space
# of neurons in each hidden layer	[64(**b**,**c**), 128, 256]
# of hidden layers	[1(**b**), 2(**c**), 3]
Buffer size	[1k, 2k(**b**,**c**), 4k]
Learning rate	[10^{-3}, 10^{-4}(**b**,**c**), 10^{-5}]
Batch size	[32, 64(**b**,**c**), 128]
Discount factor	[0.9(**c**), 0.95, 0.99(**b**)]
# of surrogate loss optimization epochs	[5, 10(**b**,**c**), 20]
Clipping parameter ϵ	[0.1, 0.2(**b**,**c**), 0.4]

Source: The author, 2023

5.4.3.1 TAC

The tabular actor-critic baseline is optimized for 500 trials on each of the three benchmark problems: mountain car, cart-pole and acrobot. Each trial is composed of 1000 epochs and the optimizer aims at achieving the best average latency at the last 500 epochs. The best performing parameters for each of these environments are indicated on Table 8 with letters **a**, **b** and **c**, respectively.

5.4.3.2 PPO

Table 9 presents the list of hyperparameters for the PPO algorithm, described in Section 5.4.2.2. The optimization process is identical to the TAC algorithm. Due to sparse rewards of the mountain car problem, the PPO algorithm was not used on this environment.

Table 10 – Hyperparameter search space for the proposed netowork

Hyperparameter	Search space
η	[10^{-5}(**a**), 10^{-4}, 10^{-3}(**b**,**c**,**d**), 10^{-2}]
η decay factor	[10^{-3}(**b**), 10^{-2}(**d**), 10^{-1}(**a**,**c**), 1]
η decay time (# of episodes)	[100(**b**,**d**), 500(**c**), 1000(**a**)]
θ_{open}	[10^{-3}(**b**), 10^{-2}(**a**,**c**), 10^{-1}(**d**)]
θ_{open} decay factor	[10^{-3}(**b**), 10^{-2}, 10^{-1}(**a**,**c**,**d**), 1]
θ_{open} decay time (# of episodes)	[100(**b**,**d**), 500, 1000(**a**,**c**)]
# of neurons in the first clustering layer	[10, 20(**c**), 50, 100(**a**,**b**,**d**)]
# of neurons in the second clustering layer	[10, 20(**c**), 50]
η_{td}	[10^{-3}(**a**), 10^{-2}(**c**,**d**), 10^{-1}(**b**)]
ϵ_{min}	[0.01(**a**,**b**,**c**,**d**), 0.05, 0.1]
ϵ decay time (# of episodes)	[100, 200, 500(**a**,**b**,**c**,**d**)]
Discount factor	[0.9, 0.95(**b**,**c**), 0.99(**a**,**d**)]
Actor learning rate	[10^{-3}, 10^{-2}, 10^{-1}(**a**,**b**,**c**,**d**)]
Critic learning rate	[10^{-3}, 10^{-2}(**c**), 10^{-1}(**a**,**b**,**d**)]
τ_a	[1, 5(**a**), 10(**c**,**d**), 20, 50(**b**)]
τ_c	[1, 5(**a**), 10(**b**,**c**,**d**), 20, 50]

Source: The author, 2023

5.4.3.3 Proposed model

The proposed model shares hyperparameters with the TAC algorithm, as the Actor-Critic part of the network is a neuromorphic implementation of this algorithm. However, a different search space is selected because the proposed Actor-Critic model does not evaluate the entire state space of a given environment. Rather, a hidden layer of the clustering network is used to group a number of individual discrete states.

Differently to the baseline algorithms above and due to the large number of adjustable parameters in the proposed model, a hybrid optimization strategy was adopted. The TPE algorithm was used alternatively with manual tuning, typically adjusting a few parameters at a time. The full set of optimized parameters is presented in Table 10, with letters **a**, **b** and **c** indicating the hyperparameters set for mountain cat, cart-pole and acrobot environments. Note that only the acrobot environment required a second clustering layer. An additional optimized configuration with a single clustering layer is indicated by letter **d**, as described in Section 5.4.5.

5.4.4 Results

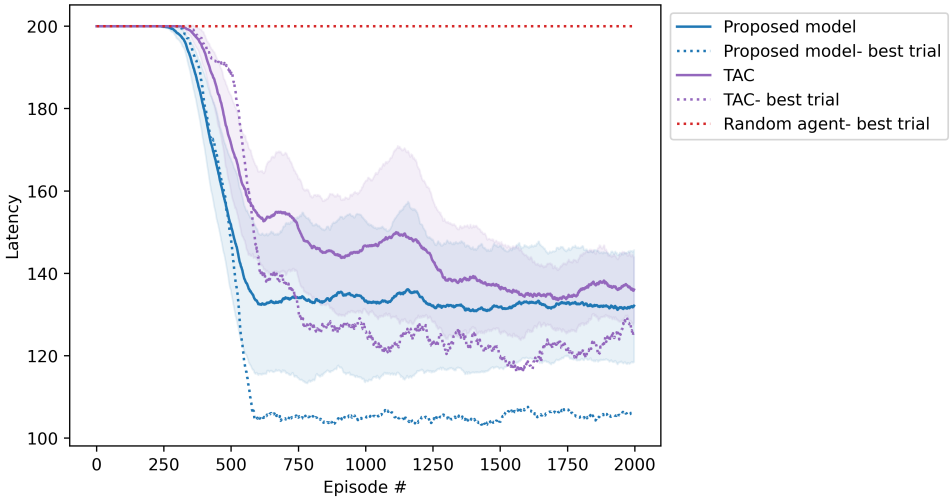
The following experimental results are obtained from ten independent runs from each combination of RL algorithms, including a random agent, and benchmark environments.

The results are presented as average of the ten trials, with respect to each episode. Additionally, the trial with the best latency is shown separately as a dotted line and the shaded region indicates standard deviation, also relative to the episode.

5.4.4.1 Mountain car

The Mountain car environment has two input dimensions and a sparse reward. The PPO algorithm did not obtain good results on this benchmark, unless the reward signal is modified to include proximity to the goal state. Otherwise, each episode terminates without positive reward. Because of this, the PPO algorithm is not included in the results presented in Figure 39. Dotted lines correspond to the best out of ten trials. Also due to the sparse reward, a random agent is unable to terminate the episode before the maximum time of 200 steps.

Figure 39 – Average latency results on the Mountain car environment. Dotted lines correspond to the best out of ten trials and the shaded regions represent the standard deviation



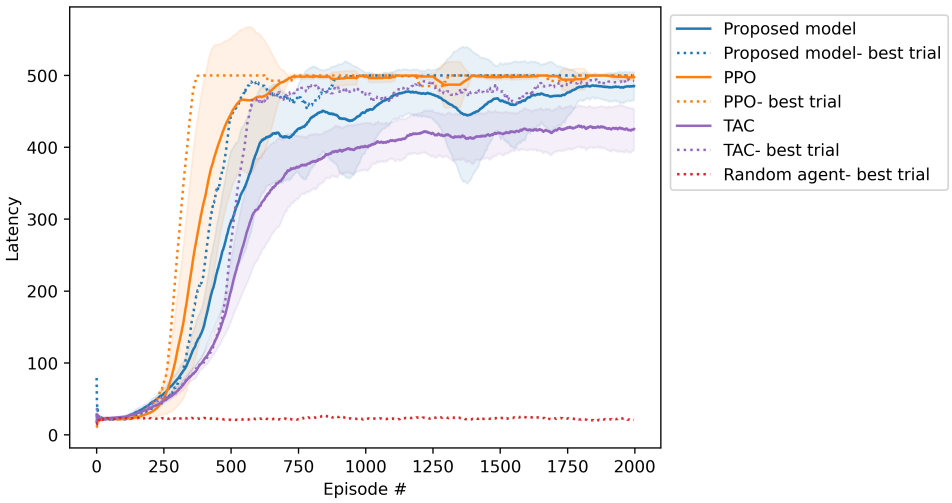
Source: The author, 2023

The proposed model and the TAC baseline present similar final latency and learning speed on average. However, a comparison of the best trials (dotted lines) suggest that the proposed network can surpass the tabular algorithm. It is also worth noting that the observation space of the actor-critic TAC in this environment consists of the entire state space, discretized using 20 bins, giving a total of 400 discrete states. Meanwhile, the proposed network employs 100 neurons for clustering the real-valued 2D input from the environment, resulting in 100 discrete states for the subsequent actor-critic component. This reduction in the state space becomes more significant in the next benchmark environments with higher state dimensions. It can also contribute to the faster learning rate observed when comparing both average and best latency curves.

5.4.4.2 Cart-pole

The proposed model is also compared to baselines on the Cart-pole environment, as illustrated in Figure 40. Dotted lines correspond to the best out of ten trials. Differently from the Mountain car problem, the reward is not sparse and is proportional to the time the agent is able to keep the pole in balance. While all three of the evaluated models are able to significantly improve the balancing time after 250 epochs, PPO provides the most stable control over the ten independent runs. The proposed model obtains a comparable performance to PPO on the best run and, as in the previous experiment, is better than TAC in terms of both average and the best runs.

Figure 40 – Average latency results on the Cart-pole environment. Dotted lines correspond to the best out of ten trials and the shaded regions represent the standard deviation



Source: The author, 2023

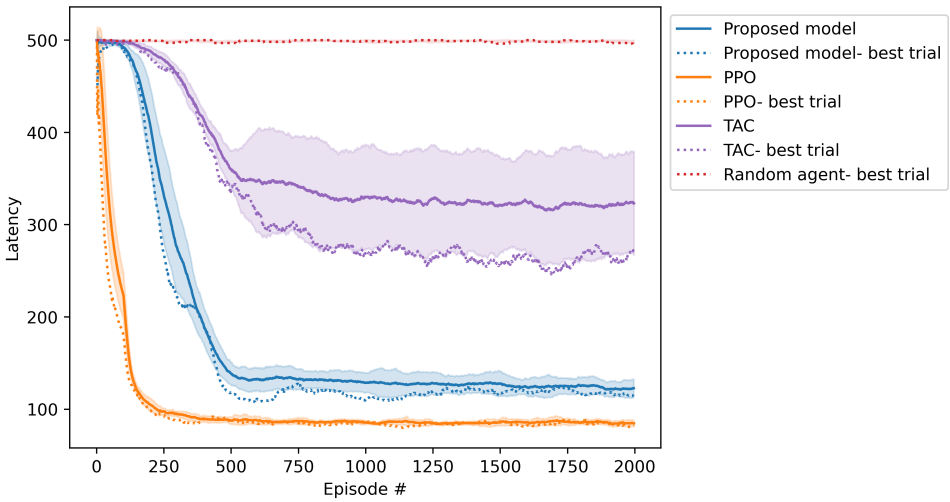
As described in Section 5.4.3.3, the proposed network uses a single clustering layer with 100 neurons, providing a reduced discrete state space for the actor-critic network. On the other hand, the TAC algorithm uses 10 bins for each input dimension, giving a total of 10^4 discrete states. While increasing the number of bins can result in an improved performance over time, this has a drawback of slower learning speed. Thus, the proposed clustering layer is able to provide enough resolution in the state space for solving the Cart-pole without requiring a very large number of neurons.

5.4.4.3 Acrobot

The Acrobot environment combines a sparse reward with additional input dimensions, making it the most challenging of the thee considered benchmarks. On this task, the proposed network is able to achieve significantly better performance by using two clustering layers. This is illustrated in Figure 41. Dotted lines correspond to the best out of ten

trials. The first layer contains 20 neurons per dimension, or a total of 120 neurons with 6 active at any given time-step. The second layer reduces this state space to just 20 neurons in total, a substantial reduction compared to 10^6 states observed by the TAC algorithm.

Figure 41 – Average latency results on the Acrobot environment. Dotted lines correspond to the best out of ten trials and the shaded regions represent the standard deviation



Source: The author, 2023

While the PPO algorithm is significantly better than the proposed one in terms of learning speed and control precision, the proposed model is still able to find a stable control strategy for the Acrobot problem. Moreover, the neuromorphic approach is fundamentally different in both memory usage and amount of changes that are applied to the network during learning. Firstly, while PPO and similar deep RL algorithms use a large memory buffer for storing previous transitions in the environment, the current approach relies on synaptic traces. Secondly, the backpropagation algorithm requires global information about the network, while in the proposed model only local computations are applied for synaptic plasticity. Thirdly, during training the synapses of the actor-critic network used by PPO are modified several times over a batch of data retrieved from the memory buffer. On the other hand, the synaptic updates on the proposed model are performed once at each time-step by using a broadcasted TD-error signal and eligibility traces.

5.4.5 Discussion and summary

An obvious advantage of the proposed network over a tabular RL algorithm is that the state space observed by the actor is drastically reduced. Table 11 provides a comparison of the proposed model and the tabular actor-critic (TAC) in terms of state space requirements. The difference becomes more significant when the number of input dimensions is increased.

Table 11 – Comparison between the proposed model and the tabular actor-critic (TAC) in terms of state space requirements

Environment	**Algorithm**	**State space**
Mountain car	Proposed	1e2
	TAC	4e2
Cart-pole	Proposed	1e2
	TAC	1e4
Acrobot	Proposed	2e1
	TAC	1e6

Source: The author, 2023

We consider the Cart-pole and Acrobot environments for an additional ablation study with the aim to evaluate the impact on performance of the individual components of the proposed network. Each experiment consists of 30 independent runs using a specific configuration of the algorithm. Average latency and standard deviation are presented and an unpaired t-test with 95% significance is used to evaluate each change in parameters.

The proposed network has two synaptic plasticity models for clustering layers: passive clustering that drives neurons to evenly represent the observed state space and TD-error modulation that encourages a denser distribution near states with higher absolute TD-error. Each of this components is individually disabled from the proposed model and evaluated on the cart-pole environment. The average latency and standard deviation are calculated from the last 1000 episodes of each run, once the a more stable control is reached. A qualitative comparison is presented in Figure 42, while the statistical significance analysis is based on data from Table 13.

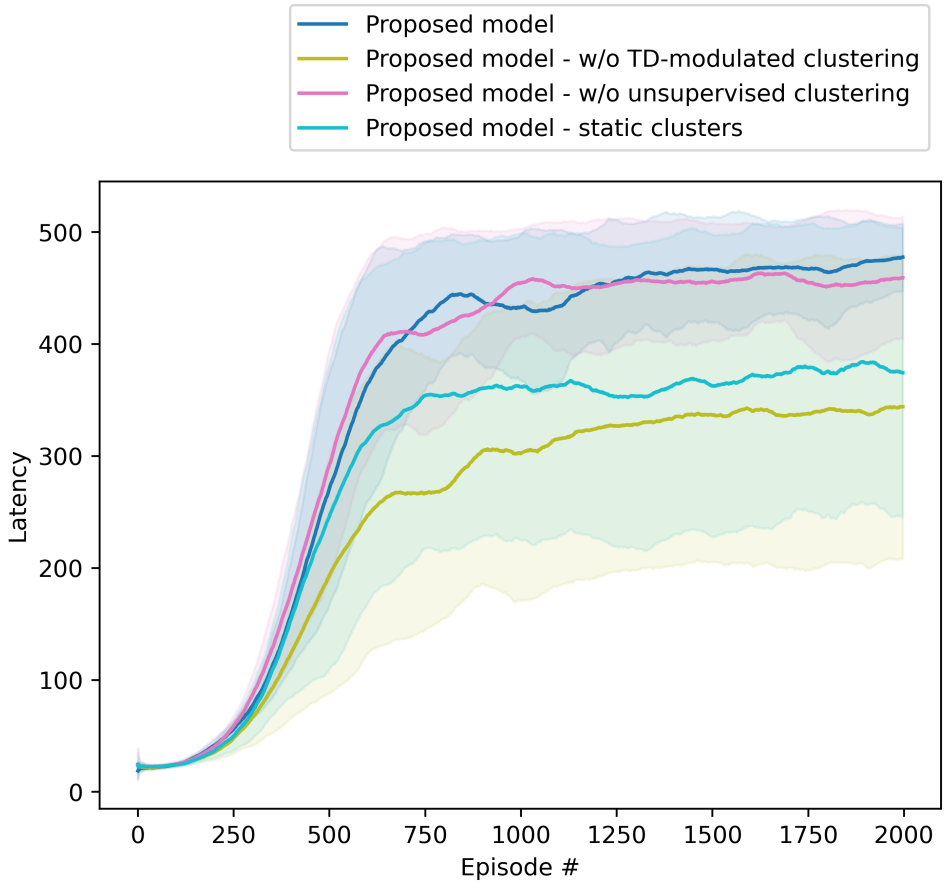
The comparison of average latencies on Table 13 suggest that TD-error modulation plays a significant role on the performance of the network. Meanwhile, there is no statistically significant difference between a configuration with and without passive clustering enabled.

Table 12 – Average and standard deviation of the latency curve for the ablation study on the cart-pole environment

Configuration	**Average latency (std)**	**Significant change?**
Proposed	460 (52)	
Proposed, w/o TD-modulated clustering	332 (134)	✓
Proposed, w/o unsupervised clustering	456 (56)	
Proposed, static clusters	367 (132)	✓

Source: The author, 2023

Figure 42 – An ablation study of the proposed model on the cart-pole environment. The shaded regions represent the standard deviation

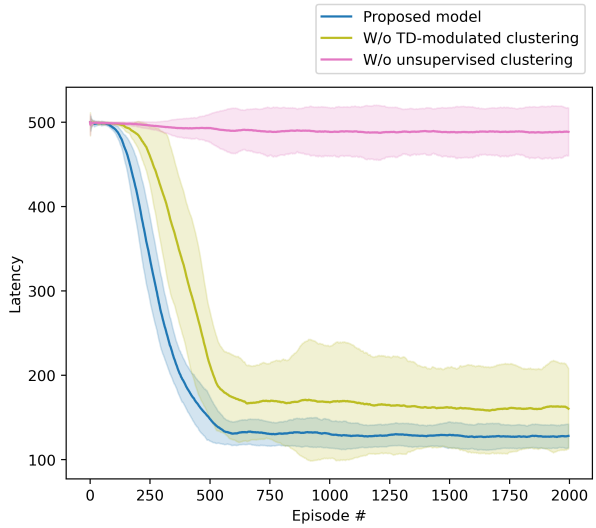


Source: The author, 2023

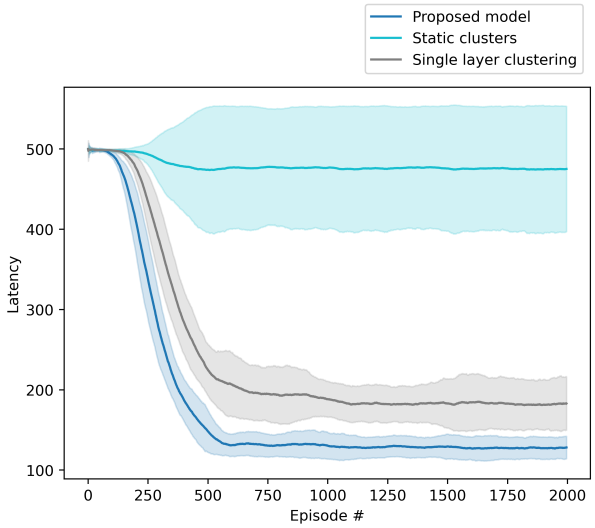
A similar ablation study on the acrobot environment is presented in Figure 43 and Table 13. However, since the best configuration of the proposed network for this environment has a second clustering layer, another configuration is optimized with a single clustering layer in order to evaluate the impact of this feature. Simply removing a layer from the proposed configuration would not result in a fair comparison, since the other hyperparameters also have to be adjusted to better fit the new configuration. An additional optimization run with 500 trials is performed, considering the same range of hyperparameters presented in Table 10.

Overall, each of the considered changes to the proposed architecture results in a statistically significant drop in performance, when compared to the initial configuration. In contrast to the previous experiment with the cart-pole environment, the unsupervised clustering plays a more significant role than the TD-modulated one. When static clusters are used, the latency increases significantly, suggesting that the dynamic nature of the clusters in the originally proposed model plays a vital role in its performance.

Figure 43 – An ablation study of the proposed model on the acrobot environment. The shaded regions represent the standard deviation.



(a)



(b)

Source: The author, 2023

Table 13 – Average and standard deviation of the latency curve for the ablation study on the acrobot environment

Configuration	**Average latency (std)**	**Significant change?**
Proposed	128 (14)	
Proposed, static clusters	475 (78)	✓
Proposed, single clustering layer	183 (29)	✓
Proposed, w/o TD-modulated clustering	163 (54)	✓
Proposed, w/o unsupervised clustering	488 (29)	✓

Source: The author, 2023

5.5 CONCLUSION AND FINAL REMARKS

This work provides a novel neuromorphic architecture aimed at solving reinforcement learning problems with real-valued observations. The proposed network contains clustering layers, based on an earlier work by (AFSHAR et al., 2020) and (BETHI et al., 2022), with an introduction of TD-error modulation and eligibility traces. The impact of the main components introduced in this architecture are evaluated in an ablation study and shown to have a significant impact on performance.

The network's effectiveness is assessed against a tabular actor-critic algorithm with eligibility traces, as well as a state-of-the-art deep learning model, Proximal Policy Optimization (PPO). The proposed model outperforms the tabular algorithm in terms of learning speed and accuracy consistently, demonstrating its capability to discover stable control policies for three control problems involving 2, 4 and 6 input dimensions. Apart from latency, other metrics can be used in future works to provide a more holistic view of the model's effectiveness. For instance, sample efficiency and memory usage are expected to further underscore the advantages of our model. Additionally, some RL problems are better described in terms of other metrics, such as cumulative reward.

While it does not surpass the PPO-based controller in terms of optimal performance, our network offers an appealing trade-off in terms of memory and hardware implementation requirements. This is because the proposed model does not require an external memory buffer and the synaptic plasticity occurs online, driven solely by local learning rules and a broadcasted TD-error signal. This aspects provide a more biologically plausible learning model, which in future may lead to more efficient algorithms in terms of computational and memory requirements. An efficient FPGA implementation of a related neuromorphic architecture, aimed at online supervised training, has been demonstrated in Mehrabi et al. (2023). This advantages in terms of hardware implementation requirements are especially relevant in edge computing or environments with limited computational resources. Finally, spiking networks are generally perceived to be more robust to loss of individual components, noise and adversarial attacks (SHARMIN et al., 2019). This characteristic is expected to be present in the proposed model and could be further evaluated in future works.

6 CONCLUSION

The goal of this thesis is to bridge the performance gap between spiking models and Deep Reinforcement Learning (DRL) algorithms on specific reinforcement learning tasks. To this end, three stages of research were undertaken, each building upon the last and improving capabilities of Spiking Neural Networks (SNNs) in the field of Reinforcement Learning (RL).

The first stage introduced a novel SNN architecture that addressed the scalability issues regarding the state space in related models. Our findings showed that this spiking model performs comparably with DRL algorithms on simple RL control tasks while offering less complexity in terms of memory and computation requirements. This research sets the stage for the two subsequent endeavors, each exploring the potential applications of SNNs in RL.

The second stage further evaluates and improves the previously proposed model, combining it with a binary feature extraction network. This binary convolutional neural network (CNN) was pre-trained on a set of naturalistic RGB images and then applied to a modified grid-world task. Improvements in architecture and dynamics are implemented to address this more challenging task with image observations. The proposed network demonstrated competitive performance with DRL algorithms, albeit in an environment with a finite set of observations. However, the use of a pretrained network for feature extraction is expected to be a limiting factor for dynamic environments, as the dependency on a pre-trained network might make the spiking actor less adaptive to changing conditions in an environment.

Given limitations in state space observations of the first two models, the third and final stage presents a neuromorphic architecture for tackling RL problems with real-valued observations. This model incorporated clustering layers, Temporal Difference (TD)-error modulation and eligibility traces. While the proposed model did not outperform PPO in optimal performance, it presented an appealing alternative in terms of computational and hardware implementation requirements without the need for an external memory buffer or global error gradient computation.

Compared to existing literature, our approach differs from most of the commonly found methods that combine spiking networks and RL. Firstly, we use a more compact state space representation than approaches involving liquid state machines (WEIDEL; DUARTE; MORRISON, 2021; TANG et al., 2021). Secondly, the proposed model does not rely on approximation of the backpropagation error and gradient descent techniques used to train traditional ANNs (BELLEC et al., 2020; AKL et al., 2023).

Although the proposed models have shown promise, they are still sensitive to parameter choices and are yet to be tested on real-world hardware implementations. Thus, future

research can focus on the following directions:

- Automatic parameter and architecture tuning – the experiments suggest that the model's performance is sensitive to the choice of several parameters. Future work could discover how best to apply evolutionary algorithms to find an architecture and a parameter set for a given problem. Automated machine learning (AutoML) is widely used for deep learning architecture search. On a smaller scale, Qiu et al. (2018) apply an evolutionary algorithm to spiking architectures for nonlinear control problems. Additionally, evolutionary algorithms such as NEAT (STANLEY; MIIKKULAINEN, 2002) have been shown to generate efficient spiking controllers for simple tasks (QIU et al., 2018). An evolutionary algorithm could be complementary to the online synaptic plasticity presented here.

- Scalability to more complex problems – the proposed model was effective in solving control problems with 2, 4 and 6 input dimensions. However, the most interesting problems involve larger state and action spaces. Future work could investigate the model's scalability to problems with even larger input dimensions. To this end, works in representation learning (OTA et al., 2020; ISLAM et al., 2022; STOOKE et al., 2021) could be studied and used as inspiration for a neuromorphic approach. Regarding action space scalability, hierarchical organization (RASMUSSEN; VOELKER; ELIASMITH, 2017) and MAP-Elites (CULLY et al., 2015) are two promising methods for learning in complex action spaces and could be combined with the proposed spiking network.

- Hardware implementation – the study highlighted the potential hardware efficiency of the proposed model. Future work could focus on the actual implementation of this architecture in hardware and investigate its performance in real-world applications.

- Population coding – the current method does not rely on population coding, which can increase the number of parameters within the network and lead to slower learning speeds. However, a more robust learning and better scalability are potential benefits of this encoding method (TANG et al., 2021). Future work could explore a hybrid approach that combines the current method with elements of population coding to potentially improve performance without significantly increasing the number of parameters.

- Spatio-temporal processing – in the current implementation, the agent is provided with enough sensory information at any time to perform the task. For example, an acrobot agent receives both the angles and the angular velocities of the joints at each time step. In future work, we plan to use additional synaptic traces and delays to learn the necessary spatio-temporal filters from raw sensory data. Processing of

temporally encoded information has been demonstrated on a related neuromorphic architecture for supervised learning (BETHI et al., 2022).

REFERENCES

ABBOTT, L. F. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain research bulletin*, Citeseer, v. 50, n. 5-6, p. 303–304, 1999.

AFSHAR, S.; RALPH, N.; XU, Y.; TAPSON, J.; SCHAIK, A. v.; COHEN, G. Event-based feature extraction using adaptive selection thresholds. *Sensors*, MDPI, v. 20, n. 6, p. 1600, 2020.

AKIBA, T.; SANO, S.; YANASE, T.; OHTA, T.; KOYAMA, M. Optuna: A next-generation hyperparameter optimization framework. In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. [S.l.: s.n.], 2019.

AKL, M.; ERGENE, D.; WALTER, F.; KNOLL, A. Toward robust and scalable deep spiking reinforcement learning. *Frontiers in Neurorobotics*, Frontiers, v. 16, p. 314, 2023.

ANDRYCHOWICZ, M.; WOLSKI, F.; RAY, A.; SCHNEIDER, J.; FONG, R.; WELINDER, P.; MCGREW, B.; TOBIN, J.; ABBEEL, P.; ZAREMBA, W. Hindsight experience replay. *arXiv preprint arXiv:1707.01495*, 2017.

ANDRYCHOWICZ, O. M.; BAKER, B.; CHOCIEJ, M.; JOZEFOWICZ, R.; MCGREW, B.; PACHOCKI, J.; PETRON, A.; PLAPPERT, M.; POWELL, G.; RAY, A. et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, SAGE Publications Sage UK: London, England, v. 39, n. 1, p. 3–20, 2020.

BALLARD, D. H.; JEHEE, J. Dynamic coding of signed quantities in cortical feedback circuits. *Frontiers in psychology*, Frontiers Research Foundation, v. 3, p. 254, 2012.

BELLEC, G.; SCHERR, F.; SUBRAMONEY, A.; HAJEK, E.; SALAJ, D.; LEGENSTEIN, R.; MAASS, W. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, Nature Publishing Group, v. 11, n. 1, p. 1–15, 2020.

BERGSTRA, J. S.; BARDENET, R.; BENGIO, Y.; KÉGL, B. Algorithms for hyper-parameter optimization. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2011. v. 24, p. 2546–2554.

BETHI, Y.; XU, Y.; COHEN, G.; SCHAIK, A. V.; AFSHAR, S. An optimized deep spiking neural network architecture without gradients. *IEEE Access*, IEEE, v. 10, p. 97912–97929, 2022.

BING, Z.; JIANG, Z.; CHENG, L.; CAI, C.; HUANG, K.; KNOLL, A. End to end learning of a multi-layered snn based on r-stdp for a target tracking snake-like robot. In: IEEE. *2019 International Conference on Robotics and Automation (ICRA)*. [S.l.], 2019. p. 9645–9651.

BING, Z.; MESCHEDE, C.; CHEN, G.; KNOLL, A.; HUANG, K. Indirect and direct training of spiking neural networks for end-to-end control of a lane-keeping vehicle. *Neural Networks*, Elsevier, v. 121, p. 21–36, 2020.

BING, Z.; MESCHEDE, C.; HUANG, K.; CHEN, G.; ROHRBEIN, F.; AKL, M.; KNOLL, A. End to end learning of spiking neural network based on r-stdp for a lane keeping vehicle. In: IEEE. *2018 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.], 2018. p. 1–8.

BING, Z.; MESCHEDE, C.; RÖHRBEIN, F.; HUANG, K.; KNOLL, A. C. A survey of robotics control based on learning-inspired spiking neural networks. *Frontiers in neurorobotics*, Frontiers, v. 12, p. 35, 2018.

BROCKMAN, G.; CHEUNG, V.; PETTERSSON, L.; SCHNEIDER, J.; SCHULMAN, J.; TANG, J.; ZAREMBA, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

CHALADZE, G.; KALATOZISHVILI, L. *Linnaeus 5 dataset for machine learning*. [S.l.], 2017.

CHEVTCHENKO, S. F.; BARBOSA, E. J.; CAVALCANTI, M. C.; AZEVEDO, G. M.; LUDERMIR, T. B. Combining ppo and incremental conductance for mppt under dynamic shading and temperature. *Applied Soft Computing*, Elsevier, v. 131, p. 109748, 2022.

CHEVTCHENKO, S. F.; BETHI, Y.; LUDERMIR, T. B.; AFSHAR, S. A neuromorphic architecture for reinforcement learning from real-valued observations. *arXiv preprint arXiv:2307.02947*, 2023.

CHEVTCHENKO, S. F.; LUDERMIR, T. B. Learning from sparse and delayed rewards with a multilayer spiking neural network. In: IEEE. *2020 International Joint Conference on Neural Networks (IJCNN)*. [S.l.], 2020. p. 1–8.

CHEVTCHENKO, S. F.; LUDERMIR, T. B. Combining stdp and binary networks for reinforcement learning from images and sparse rewards. *Neural Networks*, Elsevier, v. 144, p. 496–506, 2021.

CHUNG, S.; KOZMA, R. Reinforcement learning with feedback-modulated td-stdp. *arXiv preprint arXiv:2008.13044*, 2020.

COURBARIAUX, M.; HUBARA, I.; SOUDRY, D.; EL-YANIV, R.; BENGIO, Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

CULLY, A.; CLUNE, J.; TARAPORE, D.; MOURET, J.-B. Robots that can adapt like animals. *Nature*, Nature Publishing Group, v. 521, n. 7553, p. 503, 2015.

DHARIWAL, P.; HESSE, C.; KLIMOV, O.; NICHOL, A.; PLAPPERT, M.; RADFORD, A.; SCHULMAN, J.; SIDOR, S.; WU, Y.; ZHOKHOV, P. *OpenAI Baselines*. [S.l.]: GitHub, 2017. <https://github.com/openai/baselines>.

DIEHL, P. U.; COOK, M. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience*, Frontiers Media SA, v. 9, p. 99, 2015.

ENGSTROM, L.; ILYAS, A.; SANTURKAR, S.; TSIPRAS, D.; JANOOS, F.; RUDOLPH, L.; MADRY, A. Implementation matters in deep rl: A case study on ppo and trpo. In: *International conference on learning representations*. [S.l.: s.n.], 2019.

FELDMANN, J.; YOUNGBLOOD, N.; WRIGHT, C. D.; BHASKARAN, H.; PERNICE, W. H. P. All-optical spiking neurosynaptic networks with self-learning capabilities. *Nature*, v. 569, n. 7755, p. 208–214, 2019. ISSN 1476-4687. Disponível em: <https://doi.org/10.1038/s41586-019-1157-8>.

FLORIAN, R. V. Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation*, MIT Press, v. 19, n. 6, p. 1468–1502, 2007.

FRADY, E. P.; ORCHARD, G.; FLOREY, D.; IMAM, N.; LIU, R.; MISHRA, J.; TSE, J.; WILD, A.; SOMMER, F. T.; DAVIES, M. Neuromorphic nearest neighbor search using intel's pohoiki springs. In: *Proceedings of the Neuro-inspired Computational Elements Workshop*. [S.l.: s.n.], 2020. p. 1–10.

FRÉMAUX, N.; GERSTNER, W. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in neural circuits*, Frontiers, v. 9, p. 85, 2016.

FRÉMAUX, N.; SPREKELER, H.; GERSTNER, W. Reinforcement learning using a continuous time actor-critic framework with spiking neurons. *PLoS computational biology*, Public Library of Science, v. 9, n. 4, p. e1003024, 2013.

FRIEDRICH, J.; URBANCZIK, R.; SENN, W. Code-specific learning rules improve action selection by populations of spiking neurons. *International journal of neural systems*, World Scientific, v. 24, n. 05, p. 1450002, 2014.

GALLEGO, G.; DELBRUCK, T.; ORCHARD, G.; BARTOLOZZI, C.; TABA, B.; CENSI, A.; LEUTENEGGER, S.; DAVISON, A.; CONRADT, J.; DANIILIDIS, K. et al. Event-based vision: A survey. *arXiv preprint arXiv:1904.08405*, 2019.

GALLEGO, G.; DELBRÜCK, T.; ORCHARD, G.; BARTOLOZZI, C.; TABA, B.; CENSI, A.; LEUTENEGGER, S.; DAVISON, A. J.; CONRADT, J.; DANIILIDIS, K. et al. Event-based vision: A survey. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 44, n. 1, p. 154–180, 2020.

GERSTNER, W.; LEHMANN, M.; LIAKONI, V.; CORNEIL, D.; BREA, J. Eligibility traces and plasticity on behavioral time scales: experimental support of neohebbian three-factor learning rules. *Frontiers in neural circuits*, Frontiers Media SA, v. 12, 2018.

GU, S.; HOLLY, E.; LILLICRAP, T.; LEVINE, S. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: IEEE. *2017 IEEE international conference on robotics and automation (ICRA)*. [S.l.], 2017. p. 3389–3396.

HAESSIG, G.; MILDE, M. B.; ACEITUNO, P. V.; OUBARI, O.; KNIGHT, J. C.; SCHAIK, A. V.; BENOSMAN, R. B.; INDIVERI, G. Event-based computation for touch localization based on precise spike timing. *Frontiers in neuroscience*, Frontiers, p. 420, 2020.

HASSELT, H. V.; GUEZ, A.; SILVER, D. Deep reinforcement learning with double q-learning. In: *Thirtieth AAAI conference on artificial intelligence*. [S.l.: s.n.], 2016.

HAZAN, H.; SAUNDERS, D. J.; KHAN, H.; SANGHAVI, D. T.; SIEGELMANN, H. T.; KOZMA, R. Bindsnet: A machine learning-oriented spiking neural networks library in python. *Frontiers in neuroinformatics*, Frontiers, v. 12, p. 89, 2018.

HWANGBO, J.; LEE, J.; DOSOVITSKIY, A.; BELLICOSO, D.; TSOUNIS, V.; KOLTUN, V.; HUTTER, M. Learning agile and dynamic motor skills for legged robots. *arXiv preprint arXiv:1901.08652*, 2019.

ISLAM, R.; ZANG, H.; TOMAR, M.; DIDOLKAR, A.; ISLAM, M. M.; ARNOB, S. Y.; IQBAL, T.; LI, X.; GOYAL, A.; HEESS, N. et al. Representation learning in deep rl via discrete information bottleneck. *arXiv preprint arXiv:2212.13835*, 2022.

IZHIKEVICH, E. M. Simple model of spiking neurons. *IEEE Transactions on neural networks*, IEEE, v. 14, n. 6, p. 1569–1572, 2003.

IZHIKEVICH, E. M. Solving the distal reward problem through linkage of stdp and dopamine signaling. *Cerebral cortex*, Oxford University Press, v. 17, n. 10, p. 2443–2452, 2007.

JITSEV, J.; MORRISON, A.; TITTGEMEYER, M. Learning from positive and negative rewards in a spiking neural network model of basal ganglia. In: IEEE. *The 2012 International Joint Conference on Neural Networks (IJCNN)*. [S.l.], 2012. p. 1–8.

KAISER, J.; HOFF, M.; KONLE, A.; TIECK, J. C. V.; KAPPEL, D.; REICHARD, D.; SUBRAMONEY, A.; LEGENSTEIN, R.; ROENNAU, A.; MAASS, W. et al. Embodied synaptic plasticity with online reinforcement learning. *Frontiers in neurorobotics*, Frontiers, v. 13, p. 81, 2019.

KAPPEL, D.; LEGENSTEIN, R.; HABENSCHUSS, S.; HSIEH, M.; MAASS, W. A dynamic connectome supports the emergence of stable computational function of neural circuits through reward-based learning. *Eneuro*, Society for Neuroscience, v. 5, n. 2, 2018.

KREIS, B.; HOANG, D.; DUARTE, J. *keras-training*. [S.l.]: GitHub, 2020. <https://github.com/fastmachinelearning/keras-training>.

KUMAR, M. G.; TAN, C.; LIBEDINSKY, C.; YEN, S.-C.; TAN, A. Y.-Y. A nonlinear hidden layer enables actor-critic agents to learn multiple paired association navigation. *arXiv preprint arXiv:2106.13541*, 2021.

LAGORCE, X.; ORCHARD, G.; GALLUPPI, F.; SHI, B. E.; BENOSMAN, R. B. Hots: a hierarchy of event-based time-surfaces for pattern recognition. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 39, n. 7, p. 1346–1359, 2016.

LEGENSTEIN, R.; PECEVSKI, D.; MAASS, W. A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Comput Biol*, Public Library of Science, v. 4, n. 10, p. e1000180, 2008.

LICHTSTEINER, P.; POSCH, C.; DELBRUCK, T. A 128 $\times$ 128 120 dB 15μ s latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, IEEE, v. 43, n. 2, p. 566–576, 2008.

LILLICRAP, T. P.; HUNT, J. J.; PRITZEL, A.; HEESS, N.; EREZ, T.; TASSA, Y.; SILVER, D.; WIERSTRA, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

LUKOŠEVIČIUS, M.; JAEGER, H. Reservoir computing approaches to recurrent neural network training. *Computer science review*, Elsevier, v. 3, n. 3, p. 127–149, 2009.

MAASS, W.; NATSCHLÄGER, T.; MARKRAM, H. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, MIT Press, v. 14, n. 11, p. 2531–2560, 2002.

MAMAD, O.; STUMPP, L.; MCNAMARA, H. M.; RAMAKRISHNAN, C.; DEISSEROTH, K.; REILLY, R. B.; TSANOV, M. Place field assembly distribution encodes preferred locations. *PLoS biology*, Public Library of Science, v. 15, n. 9, p. e2002365, 2017.

MEHRABI, A.; BETHI, Y.; SCHAIK, A. van; WABNITZ, A.; AFSHAR, S. Efficient implementation of a multi-layer gradient-free online-trainable spiking neural network on fpga. *arXiv preprint arXiv:2305.19468*, 2023.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. A.; VENESS, J.; BELLEMARE, M. G.; GRAVES, A.; RIEDMILLER, M.; FIDJELAND, A. K.; OSTROVSKI, G. et al. Human-level control through deep reinforcement learning. *Nature*, Nature Publishing Group, v. 518, n. 7540, p. 529, 2015.

MOZAFARI, M.; GANJTABESH, M.; NOWZARI-DALINI, A.; MASQUELIER, T. Spyketorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron. *Frontiers in Neuroscience*, Frontiers, v. 13, p. 625, 2019.

NAKANO, T.; OTSUKA, M.; YOSHIMOTO, J.; DOYA, K. A spiking neural network model of model-free reinforcement learning with high-dimensional sensory input and perceptual ambiguity. *PloS one*, Public Library of Science, v. 10, n. 3, p. e0115620, 2015.

NEFTCI, E. O.; MOSTAFA, H.; ZENKE, F. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, IEEE, v. 36, n. 6, p. 51–63, 2019.

O'KEEFE, J.; NADEL, L. *The hippocampus as a cognitive map*. [S.l.]: Oxford: Clarendon Press, 1978.

ORCHARD, G.; MEYER, C.; ETIENNE-CUMMINGS, R.; POSCH, C.; THAKOR, N.; BENOSMAN, R. Hfirst: A temporal approach to object recognition. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 37, n. 10, p. 2028–2040, 2015.

OTA, K.; OIKI, T.; JHA, D.; MARIYAMA, T.; NIKOVSKI, D. Can increasing input dimensionality improve deep reinforcement learning? In: PMLR. *International conference on machine learning*. [S.l.], 2020. p. 7424–7433.

OTSUKA, M.; YOSHIMOTO, J.; DOYA, K. Free-energy-based reinforcement learning in a partially observable environment. In: BRUGES. *ESANN*. [S.l.], 2010.

POSCH, C.; MATOLIN, D.; WOHLGENANNT, R. A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds. *IEEE Journal of Solid-State Circuits*, IEEE, v. 46, n. 1, p. 259–275, 2010.

POTJANS, W.; DIESMANN, M.; MORRISON, A. An imperfect dopaminergic error signal can drive temporal-difference learning. *PLoS computational biology*, Public Library of Science, v. 7, n. 5, p. e1001133, 2011.

POTJANS, W.; MORRISON, A.; DIESMANN, M. A spiking neural network model of an actor-critic learning agent. *Neural computation*, MIT Press, v. 21, n. 2, p. 301–339, 2009.

QIU, H.; GARRATT, M.; HOWARD, D.; ANAVATTI, S. Evolving spiking neural networks for nonlinear control problems. In: IEEE. *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. [S.l.], 2018. p. 1367–1373.

RAFFIN, A.; HILL, A.; ERNESTUS, M.; GLEAVE, A.; KANERVISTO, A.; DORMANN, N. *Stable Baselines3*. [S.l.]: GitHub, 2019. <https://github.com/DLR-RM/stable-baselines3>.

RALPH, N.; JOUBERT, D.; JOLLEY, A.; AFSHAR, S.; TOTHILL, N.; SCHAIK, A. van; COHEN, G. Real-time event-based unsupervised feature consolidation and tracking for space situational awareness. *Frontiers in neuroscience*, Frontiers Media SA, v. 16, 2022.

RASMUSSEN, D.; VOELKER, A.; ELIASMITH, C. A neural model of hierarchical reinforcement learning. *PloS one*, Public Library of Science, v. 12, n. 7, p. e0180234, 2017.

ROSENFELD, B.; SIMEONE, O.; RAJENDRAN, B. Learning first-to-spike policies for neuromorphic control using policy gradients. In: IEEE. *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. [S.l.], 2019. p. 1–5.

RUECKERT, E.; KAPPEL, D.; TANNEBERG, D.; PECEVSKI, D.; PETERS, J. Recurrent spiking networks solve planning tasks. *Scientific reports*, Nature Publishing Group, v. 6, p. 21142, 2016.

SCHRAUWEN, B.; VERSTRAETEN, D.; CAMPENHOUT, J. V. An overview of reservoir computing: theory, applications and implementations. In: *Proceedings of the 15th european symposium on artificial neural networks. p. 471-482 2007*. [S.l.: s.n.], 2007. p. 471–482.

SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.; KLIMOV, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

SCHUMAN, C. D.; KULKARNI, S. R.; PARSA, M.; MITCHELL, J. P.; DATE, P.; KAY, B. Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science*, Nature Publishing Group US New York, v. 2, n. 1, p. 10–19, 2022.

SERRE, T.; WOLF, L.; BILESCHI, S.; RIESENHUBER, M.; POGGIO, T. Robust object recognition with cortex-like mechanisms. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 29, n. 3, p. 411–426, 2007.

SHARMIN, S.; PANDA, P.; SARWAR, S. S.; LEE, C.; PONGHIRAN, W.; ROY, K. A comprehensive analysis on adversarial robustness of spiking neural networks. In: IEEE. *2019 International Joint Conference on Neural Networks (IJCNN)*. [S.l.], 2019. p. 1–8.

STANLEY, K. O.; MIIKKULAINEN, R. Evolving neural networks through augmenting topologies. *Evolutionary computation*, MIT Press, v. 10, n. 2, p. 99–127, 2002.

STOOKE, A.; LEE, K.; ABBEEL, P.; LASKIN, M. Decoupling representation learning from reinforcement learning. In: PMLR. *International Conference on Machine Learning*. [S.l.], 2021. p. 9870–9879.

SUTTON, R. S. Learning to predict by the methods of temporal differences. *Machine learning*, Springer, v. 3, n. 1, p. 9–44, 1988.

SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. [S.l.]: MIT press, 2018.

TANG, G.; KUMAR, N.; MICHMIZOS, K. P. Reinforcement co-learning of deep and spiking neural networks for energy-efficient mapless navigation with neuromorphic hardware. *arXiv preprint arXiv:2003.01157*, 2020.

TANG, G.; KUMAR, N.; YOO, R.; MICHMIZOS, K. Deep reinforcement learning with population-coded spiking neural network for continuous control. In: PMLR. *Conference on Robot Learning*. [S.l.], 2021. p. 2016–2029.

TANNEBERG, D.; PARASCHOS, A.; PETERS, J.; RUECKERT, E. Deep spiking networks for model-based planning in humanoids. In: IEEE. *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*. [S.l.], 2016. p. 656–661.

TAVANAEI, A.; GHODRATI, M.; KHERADPISHEH, S. R.; MASQUELIER, T.; MAIDA, A. Deep learning in spiking neural networks. *Neural Networks*, Elsevier, 2018.

THAKUR, C. S. T.; MOLIN, J.; CAUWENBERGHS, G.; INDIVERI, G.; KUMAR, K.; QIAO, N.; SCHEMMEL, J.; WANG, R. M.; CHICCA, E.; HASLER, J. O. et al. Large-scale neuromorphic spiking array processors: A quest to mimic the brain. *Frontiers in neuroscience*, Frontiers, v. 12, p. 891, 2018.

VASILAKI, E.; FRÉMAUX, N.; URBANCZIK, R.; SENN, W.; GERSTNER, W. Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail. *PLoS computational biology*, Public Library of Science San Francisco, USA, v. 5, n. 12, p. e1000586, 2009.

WATKINS, C. J. C. H. Learning from delayed rewards. King's College, Cambridge, 1989.

WEIDEL, P.; DUARTE, R.; MORRISON, A. Unsupervised learning and clustered connectivity enhance reinforcement learning in spiking neural networks. *Frontiers in computational neuroscience*, Frontiers Media SA, v. 15, p. 543872, 2021.

WILSON, D. G.; DISSET, J.; CUSSAT-BLANC, S.; DUTHEN, Y.; LUGA, H. Learning aquatic locomotion with animats. In: MIT PRESS. *Artificial Life Conference Proceedings 14*. [S.l.], 2017. p. 585–592.

WUNDERLICH, T.; KUNGL, A. F.; MÜLLER, E.; HARTEL, A.; STRADMANN, Y.; AAMIR, S. A.; GRÜBL, A.; HEIMBRECHT, A.; SCHREIBER, K.; STÖCKEL, D. et al. Demonstrating advantages of neuromorphic computation: a pilot study. *Frontiers in Neuroscience*, Frontiers, v. 13, p. 260, 2019.

XU, Y.; PERERA, S.; BETHI, Y.; AFSHAR, S.; SCHAIK, A. van. *Event-driven Spectrotemporal Feature Extraction and Classification using a Silicon Cochlea Model*. [S.l.]: arXiv, 2022.

YUAN, M.; WU, X.; YAN, R.; TANG, H. Reinforcement learning in spiking neural networks with stochastic and deterministic synapses. *Neural Computation*, MIT Press, v. 31, n. 12, p. 2368–2389, 2019.

ZHANG, D.; LI, Y.; WU, S.; RASCH, M. J. Design principles of the sparse coding network and the role of "sister cells" in the olfactory system of drosophila. *Frontiers in Computational Neuroscience*, Frontiers, v. 7, p. 141, 2013.

APPENDIX A – SUPPLEMENTARY MATERIAL

A demo code with the spiking agent is provided at <https://github.com/sergio-chevtchenko/snn-binary-sample-main>. Other materials are available upon request.

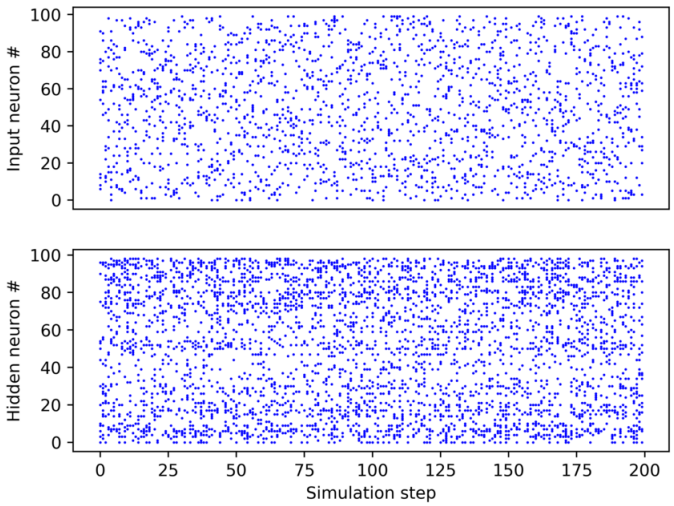
INPUT-HIDDEN CONNECTIVITY

This section presents additional experiments in order to evaluate the role of the hidden layer. The evaluated network contains only input and hidden layers, with 100 neurons each. The connectivity between these layers is encoded using the parameter set described in Section 4.3.3: [5%, 10%, 1.0, 0.25]. A random and unique pattern of 200 input signals is presented to the network, and activations of the hidden layer are recorded.

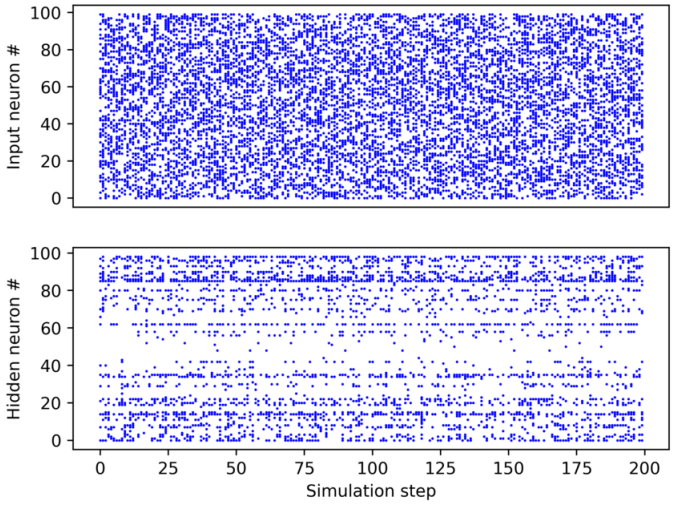
The following experiment evaluates the impact of the density of the input pattern. The density is expressed in percentage and indicates the number of randomly active inputs at any time. Sample activations of input and hidden signals are presented in Figures 44a and 44b, for a 10% and 50% input density, respectively. Jaccard distance is a common measure of dissimilarity and is used here to evaluate separation within binary patterns produced by each layer. An average of pairwise distances is computed within the recorded patterns of either input or hidden layers. This average Jaccard distance is presented in Figure 45 as a function of input density.

It is worth noting that while this experiment illustrates the role of the hidden layer, the connectivity optimization presented in Section 4.4.3.2 does not use Jaccard distance as a cost function. This is because an error rate computed from the action layer is a more direct measure of the network performance.

Figure 44 – Sample patterns produced by the input and hidden layers



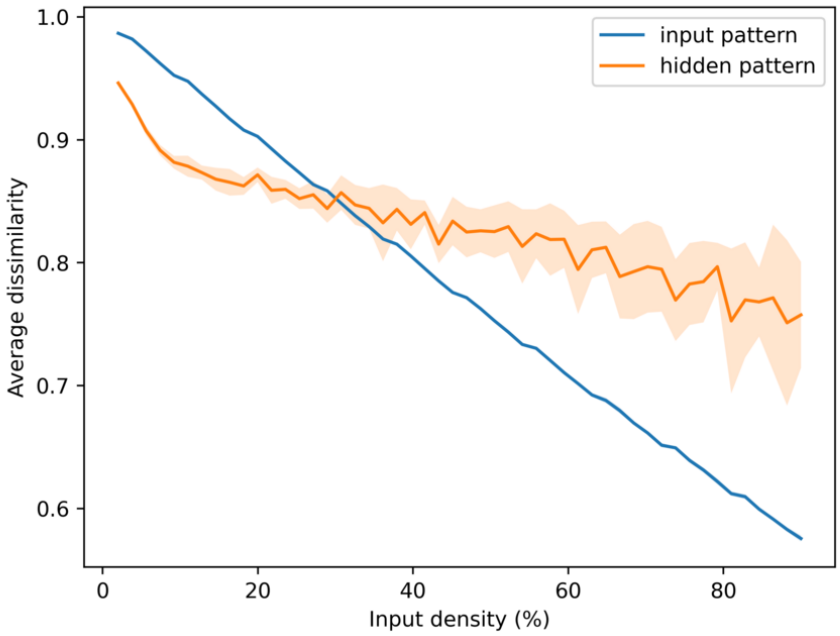
(a)



(b)

Source: The author, 2023

Figure 45 – Evaluation of patterns produced by the input and hidden layers as a function of input density



Source: The author, 2023

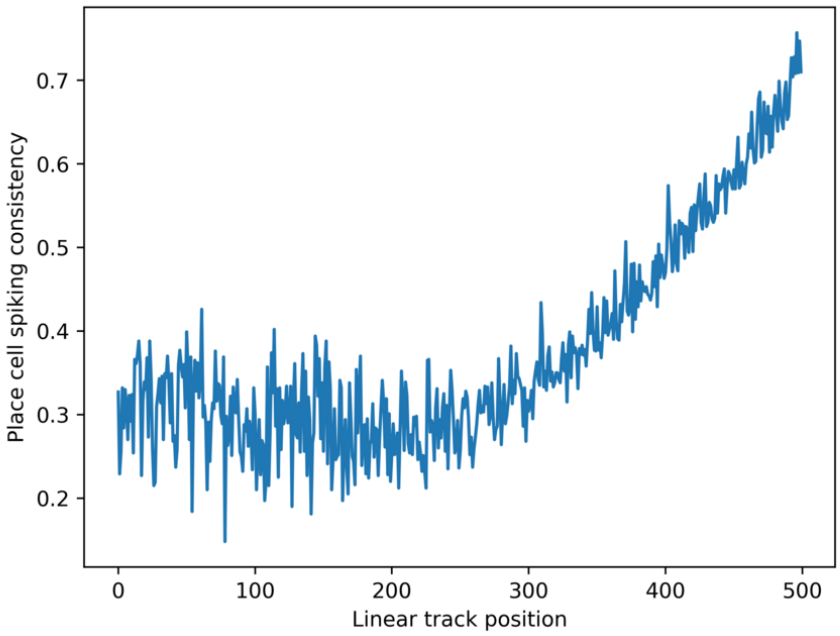
PLACE NEURONS

In the following experiment we consider the proposed spiking network with feature extraction, input, hidden and place layers. The feature extraction network receives a sequence of 500 images on each episode. Consequently, the input layer receives a sequence of 500 dense binary signals, similar to illustrated in Figure 44b. At the end of an episode, a reward signal is broadcasted to the network. The network configuration is based on the optimized configuration **b** from Table 7, except for the following adjustments due to the constant length of the episode: τ_e and τ_s are kept constant at 100 and 1000, respectively. The number of place cells is also set to 500 (the length of an episode).

Considering that the reward is consistently delivered after the same sequence of observations, we expect the place cells to also start spiking consistently after some time. The place cell spiking consistency is measured after 100 episodes for each of 500 positions at the linear track. This consistency is defined as $1 - \frac{\text{\# of place neurons that have fired at the position}}{\text{total \# of place neurons}}$.

A consistency value close to 1 means that the corresponding position is represented by a single or very few place neurons. On the other hand, a consistency close to 0 indicates that a random place neuron has fired at the given position in the last 100 episodes. The result is presented in Figure 46. Note that the consistency is highest when closer to the last position. This is due to the eligibility trace decay parameter τ_e, described in Equation 3.2.

Figure 46 – Evaluation of place cell firing consistency after 100 episodes of a linear track experiment



Source: The author, 2023

CONNECTIVITY OPTIMIZATION

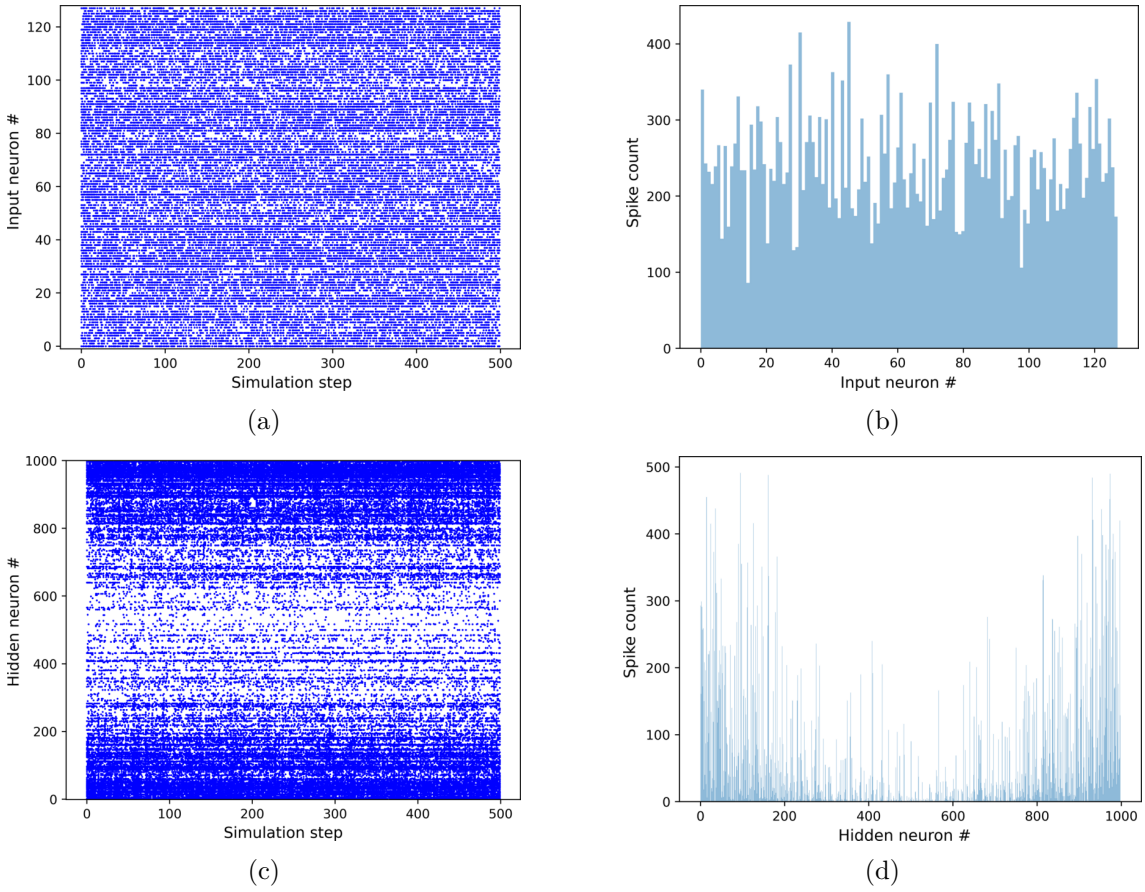
This section provides additional results from the connectivity optimization experiments described in Section 4.4.4.1. Table 14 lists the ten best connectivity configurations in terms of error rate, defined in Equation 4.2.

Table 14 – Ten best connectivity configurations

Configuration	**Error rate (std)**
[5%, 10%, 1.0, 0.25]	0.50% (0.24%)
[5%, 10%, 0.1, 0.25]	0.58% (0.23%)
[5%, 10%, 1.0, 0.5]	0.62% (0.21%)
[5%, 10%, 0.1, 1.0]	0.64% (0.23%)
[10%, 20%, 1.0, 1.0]	0.66% (0.25%)
[5%, 10%, 0.25, 0.1]	0.68% (0.31%)
[5%, 10%, 1.0, 0.1]	0.70% (0.30%)
[5%, 5%, 0.1, 0.1]	0.70% (0.16%)
[25%, 20%, 0.25, 0.25]	0.70% (0.22%)
[25%, 20%, 0.5, 0.1]	0.72% (0.32%)

Figures 47a and 47c provide sample activations from the input and hidden layers during observations from the linear track experiment. Input activations are provided by the feature extraction network. Figures 47b and 47d illustrate the spike count per neuron of each layer during the experiment.

Figure 47 – A sample of activation patterns from the linear track experiment



Source: The author, 2023

PPO WITH BINARYNET

Note that the baseline algorithms use a CNN followed by a fully connected layer as a policy network. Thus, it is reasonable to suppose that the use of a pre-trained BinaryNet instead of a CNN as a feature extractor may provide a biased advantage to the proposed spiking network.

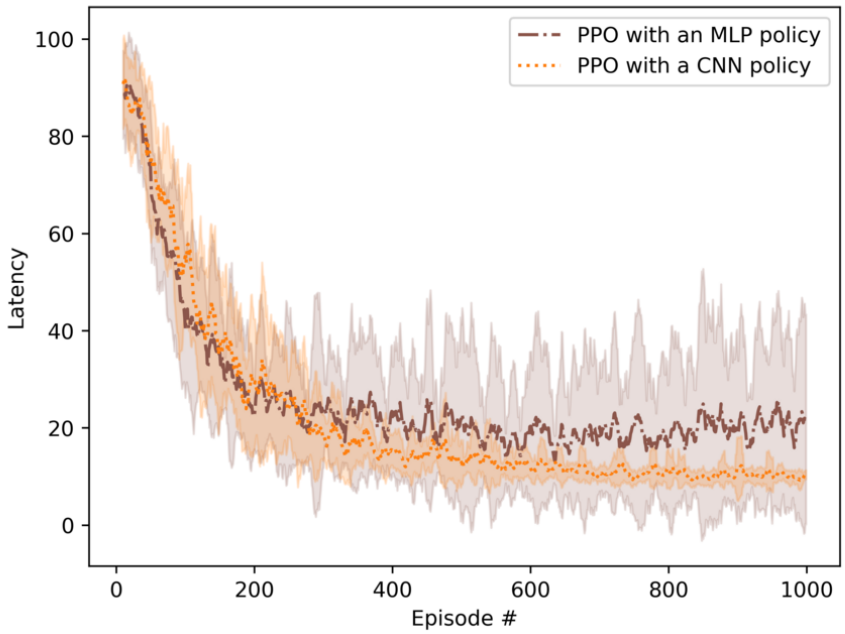
In order to test this, the PPO algorithm is optimized with observations from the BinaryNet, in the same manner as the spiking model. The optimization is the same as described in Section 4.4.3.1. The list of optimized hyperparameters is based on Table 9, except for the convolutional layer, which is replaced by an MLP. The new set of hyperparameters is provided in Table 15, and the highlighted parameters are used in this experiment.

Table 15 – Optimized hyperparameters of PPO

Hyperparameter	Search space
# of hidden layers	[**1**, 2]
# of neurons in the hidden layers	[**32**, 64, 128]
Buffer size	[**1k**, 2k, 4k]
Learning rate	[$\mathbf{10^{-3}}$, 10^{-4}, 10^{-5}]
Batch size	[32, 64, **128**]
Discount factor	[0.9, **0.99**, 0.999]
# of surrogate loss optimization epochs	[5, **10**, 20]
Clipping parameter	[0.1, 0.2, **0.4**]

A comparison of optimized versions of PPO with CNN and BinaryNet is presented in Figure 48. The results suggest that the BynaryNet does not provide an advantage in performance for the PPO algorithm. An investigation of whether the use of a spiking CNN with online training would improve performance of our network is left for future works.

Figure 48 – Comparison of optimized version of the PPO algorithm with CNN and BinaryNet as feature extractors. Latency is averaged across ten trials with shaded region corresponding to standard deviation



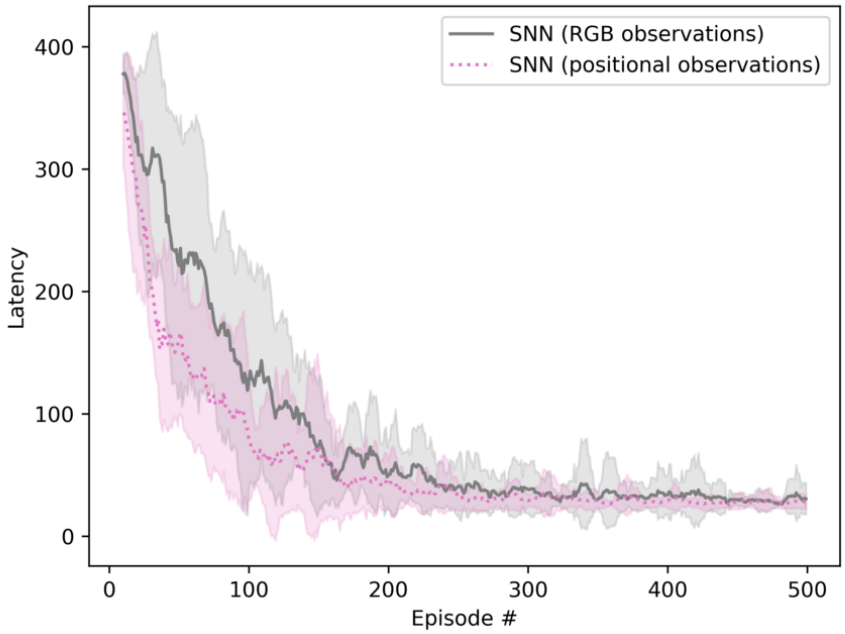
Source: The author, 2023

GRID-WORLD WITH POSITIONAL OBSERVATIONS

The grid-world environment evaluated in the present work provides observations in form of naturalistic RGB images, as described in Section 4.4.1. The dimensionality of this input is reduced by a BinaryNet feature extractor and the spiking network receives a dense 128-bit signal, as illustrated in Figure 19. On the other hand, positional observations can be encoded with a sparse binary vector. In the case of a 20×20 grid-world, the observations would consist of a 40-bit vector with two active bits at each time.

In this experiment we compare the two environment observations by using the network configuration **b** from Table 7. Note that this network is only optimized for image observations but is used to illustrate that very sparse positional observations are less challenging. The latency curve from the experiment is presented in Figure 49.

Figure 49 – The network configuration **b** from Table 7 is used to compare two 20×20 grid-world environments with positional and RGB observations



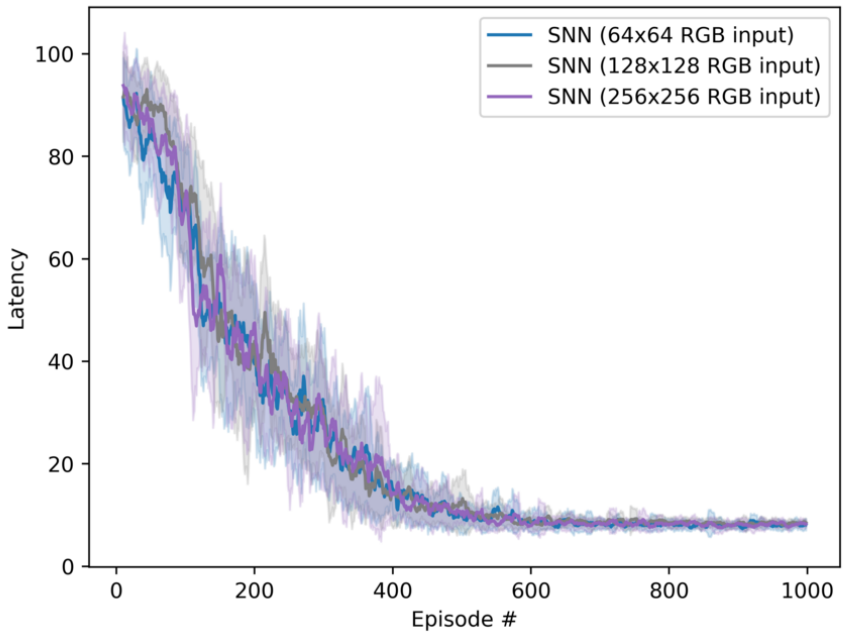
Source: The author, 2023

SCALABILITY IN TERMS OF IMAGE SIZE

This experiment evaluates the scalability of the feature extraction network (BinaryNet) in terms of input image size. We use a 10×10 grid-world with observations expanded to 128×128 and 256×256 pixels. The latency curves are provided in Figure 50. As in all

three configurations, the spiking agent receives the same 128 bit feature vector and its performance is not affected by the input image size.

Figure 50 – Evaluation of the scalability of the feature extraction network (BinaryNet) in terms of input image size



Source: The author, 2023