Learning from Sparse and Delayed Rewards with a Multilayer Spiking Neural Network

Sérgio F. Chevtchenko, Teresa B. Ludermir
Centro de Informática - CIn
Universidade Federal de Pernambuco - UFPE
{sfc, tbl}@cin.ufpe.br

***Abstract*—Biological brains still far exceed artificial intelligence systems, both in terms of control capabilities and power consumption. Spiking neural networks (SNNs) are a promising model, inspired by neuroscience and functionally closer to the way neurons process information. While recent advancements in neuromorphic hardware allow energy efficient synthesis of spiking networks, the training of such networks remains an open problem. In this work we focus on reinforcement learning with sparse and delayed rewards. The proposed architecture has four distinct layers and addresses the limitation of previous models in terms of scalability with input dimensions. Our SNN is evaluated on classical reinforcement learning and control tasks and is compared to two common RL algorithms: Q-learning and deep Q-network (DQN). Experiments demonstrate that the proposed network outperforms Q-learning on a task with six-dimensional observation space and compares favorably to the evaluated DQN configurations in terms of stability and memory requirements.**

***Index Terms*—Reinforcement learning, spiking neural networks, reward-modulated STDP**

I. Introduction

Reinforcement learning (RL) closely resembles, and is inspired by, the way animals learn [1]. It is a very flexible paradigm that can be used to train an agent to perform a task without previous knowledge of how to execute it step by step — a reward by the end of a successful trial is enough. Recent advances in RL include algorithms that are able to play video games better than humans [2], as well as learning agile and dynamic motor skills for legged robots [3].

Despite great advancements in robotics and machine learning, artificial systems still have significant disadvantages when compared to their biological counterparts. First, while animals can quickly adapt and learn new behaviors, training an artificial neural network (ANNs) is time-consuming and typically requires a large collection of examples. Secondly, inference on an already trained ANN is power-demanding and can considerably reduce autonomy on mobile applications with a limited power source, such as space exploration, robotics and wearable devices [4]. For instance, while the human brain consumes about 20 watts, the Human Brain Project's simulation of the cortex is expected to consume 500 MW, which is roughly equivalent to 250 thousand households [5], [6].

Traditional ANNs use neural models that receive and transmit continuous signals, essentially working as universal function approximators. On the other hand, biological neurons send and receive information via discrete spikes. Based on insights from neuroscience, the artificial spiking neural network (SNN) is a promising alternative to leverage both spatio-temporal information processing and low energy demand. State-of-the-art low power neuromorphic hardware is able to simulate between 10^5 to 10^6 spiking neurons in real time [6]. A notable recent work proposes and implements a fully optical spiking neural network, potentially further increasing bandwidth and processing speed of SNNs [7].

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 [2]. While it is possible to train SNNs using modified gradient-based optimization, this does not leverage the low-power requirements of biological neurons [8]. Instead of computing a global gradient over all of the synapses, SNNs can employ a biologically plausible process called spike-timing-dependent plasticity (STDP). This synaptic plasticity rule only requires each synapse to be aware of corresponding pre- and post-synaptic neurons. STDP is illustrated in Figure 1 and can be applied to a network of neurons. 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 provided with a food reward after a successful task completion.

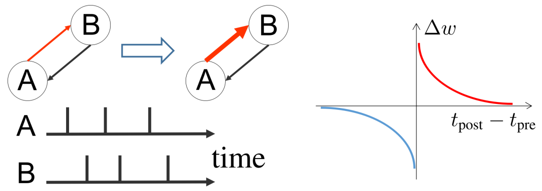

Fig. 1: Illustration of the spike-timing-dependent plasticity (STDP). Neuron A repeatedly causes neuron B to fire, after a small delay. Over time, the connection from neuron A to B is strengthened, as a function of the delay between spikes.

An analysis of related works, presented in Section II, 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. Our model is intended for future implementations on hardware. As such, we follow a strategy similar to recent works [9], [10] and use simplified neural and synaptic plasticity models, prioritizing compactness over biological realism.

II. Related Work

Early versions of SNNs with reward-modulated plasticity are independently proposed by Izhikevich [11] and Florian [12]. In both instances, eligibility traces and STDP are employed. In particular, Florian [12] demonstrates that a fully connected multilayered network of spiking neurons can solve a temporally coded XOR problem with delayed reward.

Portjans et al. [13] 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 and Gerstner [14] 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 [15]. 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 [13] and [14] 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, [14] 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. [16] 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. [17] 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 dimentions. A related, but more scalable learning rule through imitation is introduced by Tannenberg et al. [18]. The present work takes inspiration from the models proposed by [17] [18], while maintaining the reinforcement learning paradigm and introducing a population coding that can be scaled well to more than four dimensions.

Wilson et al. [19] 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 [11]. 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. [20] 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 [21], 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.

The power consumption advantage of neuromorphic hardware is demonstrated by Wunderlich et al. [22], 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. [23]. This rules 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.

III. Proposed Spiking Network

A diagram of the proposed SNN is shown in Figure 2. 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. The following sections describe in more details the neural model and the structure of the proposed network.

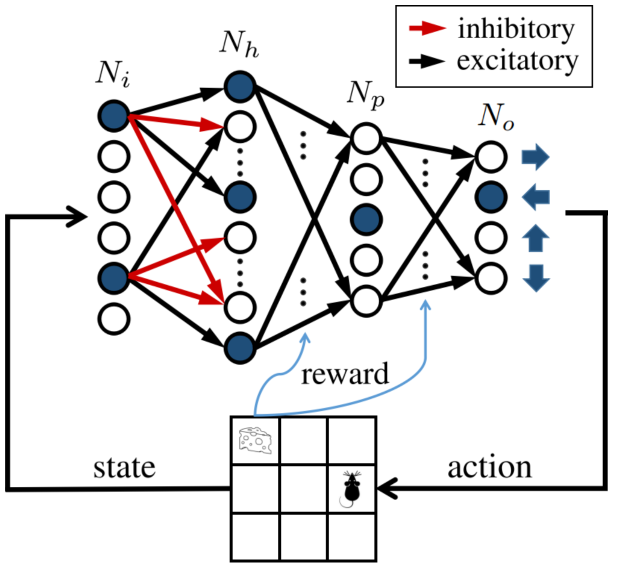

Fig. 2: Diagram of the proposed spiking network. In this setup there are three neurons per dimension ($n_d = 3$) and the agent chooses between four possible actions. For legibility, only the connections from active input neurons are shown.

A. Neural Model

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

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

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. 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.

B. Synaptic Plasticity

The following is a simplified version of the reward-modulated STDP plasticity rule introduced by Florian [12]. When postsynaptic neuron i fires, the eligibility trace $Z_{j,i}$ is set to the value of the presynaptic trace P_-, decaying exponentially with a time constant τ_e:

$$Z_{j,i}(t) = Z_{j,i}(t-1) - \frac{Z_{j,i}(t-1)}{\tau_e}. \tag{2}$$

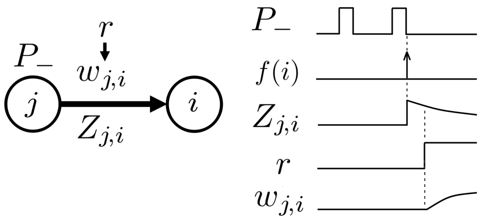

Fig. 3: The synaptic plasticity model. Presynaptic trace P_- is used to indicate that neuron j has produced a spike on the previous time step. Spiking of the postsynaptic neuron i triggers an update of the eligibility trace $Z_{j,i}$. Subsequent broadcast of the reward signal r prompts an update to the synaptic efficacy $w_{j,i}$

The change in synaptic strength between two neurons depends on presynaptic and postsynaptic spike times, as well as on the reward signal:

$$w_{j,i}(t) = w_{j,i}(t-1) - \frac{w_{j,i}(t-1)}{\tau_s} + \alpha\ r(t)\ Z_{j,i}(t), \tag{3}$$

where τ_s is the synaptic tag discharge time constant and α is the learning rate. The reward signal $r(t)$ is broadcasted to all plastic synapses. The above rule is illustrated in Figure 3. Note that the time constants τ_e and τ_s are adjusted according to the duration of a trial for a specific task.

C. 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 2 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.

D. Place Neurons

This layer is inspired by the neuronal activity of the hippocampus and represents the position of the agent in the sensory space [24]. 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 [25] 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, 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 [14] or a global max-pooling layer in [9] and increases the robustness of the learning process.

E. Output Layer

The output layer in the proposed architecture is very similar to previous works, such as [13], [15], [20] 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 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. 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 IV use a constant 2% random action probability.

IV. Experiments

A. 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.

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 [13] and [14]. This environment is illustrated in Figure 4. 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.

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.

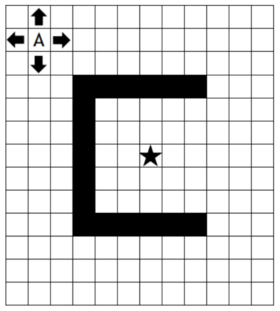

Fig. 4: Illustration of the maze task. Only the state in the center is rewarded (+1). The agent (A) starts in the upper-left corner and can move in four cardinal directions. Collisions with the U-shaped wall or with the borders of the maze are not penalized but will make the agent return to the previous position.

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 5 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. 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 [26], except for the reward policy, which is modified to be binary and sparse.

B. Baseline Models

As seen in Section II, 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 [27], a classical RL algorithm, as well as its more recent variation, Deep Q Network (DQN) [2], both described below. 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.

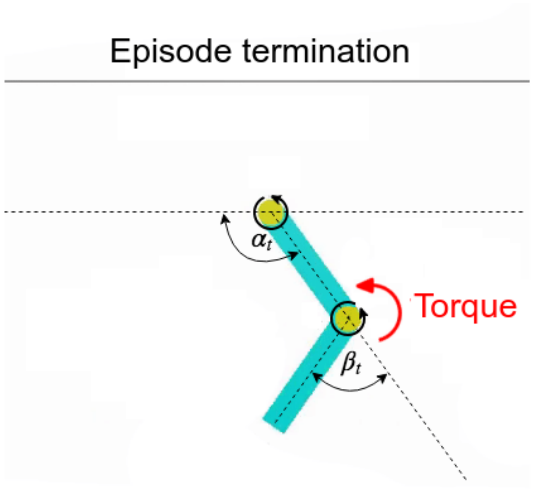

Fig. 5: Illustration of the acrobot task.

1) Q-learning: this is a classical model-free RL algorithm, proposed by Watkins [27], itself derived from temporal difference (TD) learning [28]. 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:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r + \gamma Q_{\max}(s_{t+1}) - Q(s_t, a_t)), \tag{4}$$

where α 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:

$$a_t = \begin{cases} \arg\max_i \, Q(s_t, a_i), & \text{if } R > \epsilon \\ \textit{Random}, & \text{otherwise}, \end{cases} \tag{5}$$

where R is a random number and 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.

2) DQN: the traditional Q-learning algorithm works well with a limited number of states and actions, but as demonstrated in Section IV-D, 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 4 becomes:

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta}_t + \alpha(Y_t^Q - Q(s_t, a_t; \boldsymbol{\theta}_t))\nabla_{\boldsymbol{\theta}_t} Q(s_t, a_t; \boldsymbol{\theta}_t) \tag{6}$$

$$Y_t^Q = r + \gamma Q_{\max}(s_{t+1}; \boldsymbol{\theta}_t), \tag{7}$$

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. [2], 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 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.

C. 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 [29]. 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. [29]. 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 6 and 8).

The hyperparameter search spaces for the Q-learning and DQN algorithms are described in Tables I and II. 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.

TABLE I: Q-learning hyperparameters search space

Hyperparameter	Search Space
Learning rate α	[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]

TABLE II: DQN hyperparameters search space.

Hyperparameter	Search Space
Learning rate α	[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]

The optimization algorithm is executed for a minimum of 100 trials and the evaluation 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 α – 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 [30], shown in parentheses:

- Learning rate α – 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)
- Update period τ_a – 500 (1000)

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

TABLE III: 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

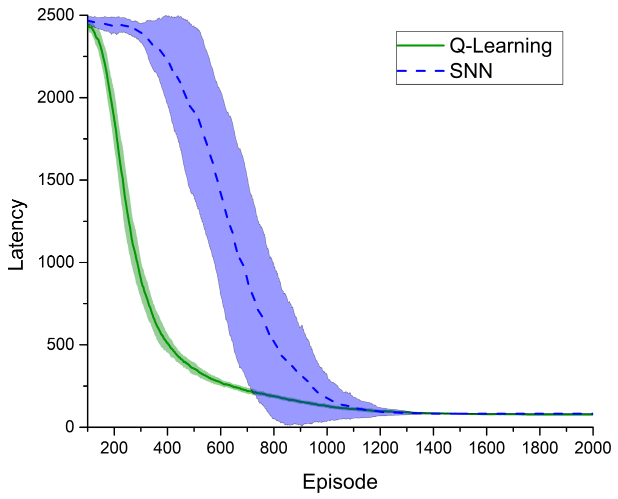

Fig. 6: Average latency on the maze task. Shaded region is the standard deviation over 10 trials.

D. Results

The proposed spiking network and the Q-learning algorithm are evaluated on the 50×50 maze task, described in Section IV-A1. We measure the latency of each agent — the number of states it takes to reach the target. Figure 6 shows the average latency as a function of episodes. 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 6, 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 [13] [14]. 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 7.

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 IV-A2. 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 8. As

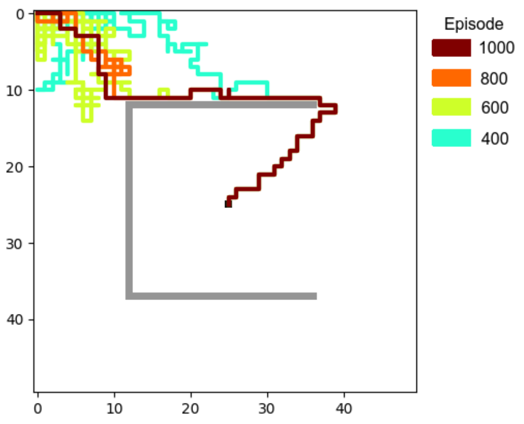

Fig. 7: Sample paths followed by an SNN on the maze task. Color codes correspond to episodes from the same trial.

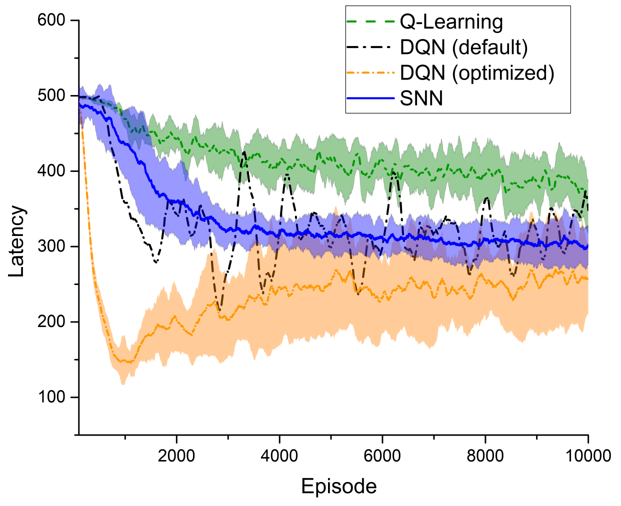

Fig. 8: Average latency on the acrobot task. Shaded region is the standard deviation over 10 trials. Standard deviation is high for the default DQN and is not shown for readability.

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 5. As this is a more difficult task, a trial lasts for 10,000 episodes.

An analysis of Figure 8 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 9.

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.

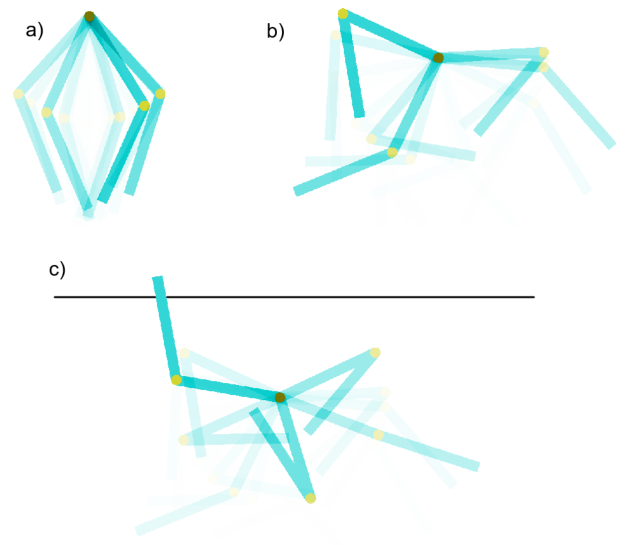

Fig. 9: Time-lapse of a successful episode with the SNN controller. The agent reaches the goal height in the final step. A video sample is available at https://youtu.be/r5BZMZ4hb_o.

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.

V. Conclusion and Future Work

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 [13]–[15], [19]. 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. It is also worth noting that our network is trained through simplified STDP instead of gradient descent, which greatly improves potential energy requirements [22].

Future endeavors could focus on improving some aspects of the proposed architecture:

- Action space scalability – most interesting problems also involve large action spaces. Hierarchical organization [31] and MAP-Elites [32] are two promising methods for learning in complex action spaces and could be combined with the proposed spiking network.
- Place neurons – the current plasticity rule between hidden and place layers could be improved to make more efficient use of the neurons. For instance, similar states that do not require a distinct action could gradually converge on a single place neuron.

- Spatio-temporal processing – in the current implementation, the agent is provided at any time with enough sensory information to perform the task. For example, an acrobot agent receives both angles and angular velocities of the joints at each time step. In future work, we plan to use additional synaptic traces and delays in order to learn the necessary spatio-temporal filters from raw sensory data.
- Hyperparameter optimization – a multiobjective optimization of hyperparameters would be an interesting research direction, leveraging, for example, learning performance and model complexity. Additionally, evolutionary algorithms such as NEAT [33] have been shown to generate efficient spiking controllers for simple tasks [34]. An evolutionary algorithm could be complementary to the online synaptic plasticity presented here.

Acknowledgment

The authors would like to thank FACEPE, CNPq and CAPES (Brazilian Research Agencies) for their financial support.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[3] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *arXiv preprint arXiv:1901.08652*, 2019.

[4] Z. Bing, C. Meschede, F. Röhrbein, K. Huang, and A. C. Knoll, "A survey of robotics control based on learning-inspired spiking neural networks," *Frontiers in neurorobotics*, vol. 12, p. 35, 2018.

[5] M. E. Raichle and D. A. Gusnard, "Appraising the brain's energy budget," *Proceedings of the National Academy of Sciences*, vol. 99, no. 16, pp. 10 237–10 239, 2002.

[6] C. S. T. Thakur, J. Molin, G. Cauwenberghs, G. Indiveri, K. Kumar, N. Qiao, J. Schemmel, R. M. Wang, E. Chicca, J. Olson Hasler *et al.*, "Large-scale neuromorphic spiking array processors: A quest to mimic the brain," *Frontiers in neuroscience*, vol. 12, p. 891, 2018.

[7] J. Feldmann, N. Youngblood, C. D. Wright, H. Bhaskaran, and W. H. P. Pernice, "All-optical spiking neurosynaptic networks with self-learning capabilities," *Nature*, vol. 569, no. 7755, pp. 208–214, 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-1157-8

[8] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Networks*, 2018.

[9] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, and T. Masquelier, "Spyketorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron," *Frontiers in Neuroscience*, vol. 13, p. 625, 2019.

[10] H. Hazan, D. J. Saunders, H. Khan, D. T. Sanghavi, H. T. Siegelmann, and R. Kozma, "Bindsnet: A machine learning-oriented spiking neural networks library in python," *Frontiers in neuroinformatics*, vol. 12, p. 89, 2018.

[11] E. M. Izhikevich, "Solving the distal reward problem through linkage of stdp and dopamine signaling," *Cerebral cortex*, vol. 17, no. 10, pp. 2443–2452, 2007.

[12] R. V. Florian, "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity," *Neural Computation*, vol. 19, no. 6, pp. 1468–1502, 2007.

[13] W. Potjans, M. Diesmann, and A. Morrison, "An imperfect dopaminergic error signal can drive temporal-difference learning," *PLoS computational biology*, vol. 7, no. 5, p. e1001133, 2011.

[14] N. Frémaux, H. Sprekeler, and W. Gerstner, "Reinforcement learning using a continuous time actor-critic framework with spiking neurons," *PLoS computational biology*, vol. 9, no. 4, p. e1003024, 2013.

[15] N. Frémaux and W. Gerstner, "Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules," *Frontiers in neural circuits*, vol. 9, p. 85, 2016.

[16] T. Nakano, M. Otsuka, J. Yoshimoto, and K. Doya, "A spiking neural network model of model-free reinforcement learning with high-dimensional sensory input and perceptual ambiguity," *PloS one*, vol. 10, no. 3, p. e0115620, 2015.

[17] E. Rueckert, D. Kappel, D. Tanneberg, D. Pecevski, and J. Peters, "Recurrent spiking networks solve planning tasks," *Scientific reports*, vol. 6, p. 21142, 2016.

[18] D. Tanneberg, A. Paraschos, J. Peters, and E. Rueckert, "Deep spiking networks for model-based planning in humanoids," in *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*. IEEE, 2016, pp. 656–661.

[19] D. G. Wilson, J. Disset, S. Cussat-Blanc, Y. Duthen, and H. Luga, "Learning aquatic locomotion with animats," in *Artificial Life Conference Proceedings 14*. MIT Press, 2017, pp. 585–592.

[20] Z. Bing, C. Meschede, K. Huang, G. Chen, F. Rohrbein, M. Akl, and A. Knoll, "End to end learning of spiking neural network based on r-stdp for a lane keeping vehicle," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

[21] Z. Bing, Z. Jiang, L. Cheng, C. Cai, K. Huang, and A. Knoll, "End to end learning of a multi-layered snn based on r-stdp for a target tracking snake-like robot," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 9645–9651.

[22] T. Wunderlich, A. F. Kungl, E. Müller, A. Hartel, Y. Stradmann, S. A. Aamir, A. Grübl, A. Heimbrecht, K. Schreiber, D. Stöckel *et al.*, "Demonstrating advantages of neuromorphic computation: a pilot study," *Frontiers in Neuroscience*, vol. 13, p. 260, 2019.

[23] M. Yuan, X. Wu, R. Yan, and H. Tang, "Reinforcement learning in spiking neural networks with stochastic and deterministic synapses," *Neural Computation*, vol. 31, no. 12, pp. 2368–2389, 2019.

[24] J. O'keefe and L. Nadel, *The hippocampus as a cognitive map*. Oxford: Clarendon Press, 1978.

[25] O. Mamad, L. Stumpp, H. M. McNamara, C. Ramakrishnan, K. Deisseroth, R. B. Reilly, and M. Tsanov, "Place field assembly distribution encodes preferred locations," *PLoS biology*, vol. 15, no. 9, p. e2002365, 2017.

[26] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[27] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[28] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[29] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, 2011, pp. 2546–2554.

[30] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017.

[31] D. Rasmussen, A. Voelker, and C. Eliasmith, "A neural model of hierarchical reinforcement learning," *PloS one*, vol. 12, no. 7, p. e0180234, 2017.

[32] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, p. 503, 2015.

[33] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[34] H. Qiu, M. Garratt, D. Howard, and S. Anavatti, "Evolving spiking neural networks for nonlinear control problems," in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2018, pp. 1367–1373.