Closed-Loop Q-Learning Control with Spiking Neuromorphic Network

Giovanni Michel, Steven Nesbit and Andrew Sornborger

January 6, 2025

Closed-loop Q-Learning Control with Spiking Neuromorphic Network

1st Giovanni T. Michel
Information Sciences, CCS-3
Los Alamos National Laboratory
Los Alamos, USA
gmichel@lanl.gov

2nd Steven C. Nesbit
Information Sciences, CCS-3
Los Alamos National Laboratory
Los Alamos, USA
nesbitsc@lanl.gov

3rd Andrew T. Sornborger
Information Sciences, CCS-3
Los Alamos National Laboratory
Los Alamos, USA
sornborg@lanl.gov

***Abstract*—Neuromorphic processors offer a promising, low-power alternative to standard von Neumann architectures. For these processors to be effectively used, it is important to enable end-to-end processing entirely on-chip to avoid the dominant power consumption of standard computational components. For this reason, control problems in autonomous agents present a compelling domain for applying neuromorphic solutions. In this simulation study, we introduce a closed-loop, spiking implementation of Q-learning. Here, we study a proof-of-principle problem: cartpole balancing. Our approach uses the OpenAI Gym for off-chip, in-the-loop simulation of cartpole dynamics. Unlike our previous work in which the Q-learning matrix was learned entirely off-chip, then transferred on-chip for testing, we now showcase an entirely spike-based training implemented in Intel's Lava Software Framework – software for neuromorphic simulation. We show that the agent can learn to balance the cartpole well after training. Our spiking implementation is a first step towards full, on-chip Q-learning.**

***Index Terms*—Reinforcement learning, spiking neural networks, hebbian learning, q-learning control, synfire-gated synfire chains, sparse coding**

I. Introduction

Reinforcement learning (RL) is a powerful method for solving complex control problems [1], but traditional implementations, often based on von Neumann architectures, are typically energy-intensive and unsuitable for real-time applications [2]. Neuromorphic computing systems, inspired by biological neural circuits, present a promising alternative for low-power, low-latency, on-chip learning [3], [4]. These systems offer significant energy efficiency and the potential for massive parallelism, making them attractive for implementing RL in resource-constrained environments.

Despite their promise, adapting traditional machine learning algorithms, such as RL, to neuromorphic architectures remains a significant challenge [5], [6]. The core difficulty lies in managing information propagation within spiking neural networks (SNNs), which serve as the foundation of neuromorphic systems [7], [8]. Unlike traditional architectures, SNNs transmit information through discrete spikes, making it difficult to control when and where spikes occur, thus making it challenging to implement sophisticated algorithms like RL.

To address this challenge, we propose a novel neuromorphic reinforcement learning framework based on synfire-gated synfire chains (SGSCs) [7], [8]. This framework enables precise control over spiking information flow within neural circuits, providing a pathway for implementing RL on neuromorphic hardware. We applied this approach to the classic cartpole balancing task, where the goal is to teach an agent to balance an inverted pendulum. Using a modular neuromorphic system integrated with the OpenAI Gym [9] simulation environment, we successfully demonstrated closed-loop control and learning.

Our results highlight the capability of SGSCs to solve complex tasks in self-contained, spike-based neural circuits by governing the propagation of spiking information. The SGSC framework not only controls the spatial propagation of spikes but also their temporal arrival at specific locations [4]. In our neuromorphic RL implementation, SGSCs are used to implement Hebbian learning updates for the Q-learning algorithm. The agent maximizes the expected reward across discrete states, with learning driven by reward signals from OpenAI Gym.

The circuit that we designed computes synaptic weight updates using the Q-learning update rule [10], providing a proof-of-concept for learning RL policies within spiking neural networks. Our work demonstrates the feasibility of implementing RL in neuromorphic systems using simulation software and lays the groundwork for deploying these circuits on neuromorphic hardware, such as Loihi 2.

A. Related Work

Previous attempts at solving the cartpole problem have explored hybrid training methods that leverage neuromorphic-inspired algorithms, such as Reward-Modulated Spike-Timing-Dependent Plasticity (R-STDP) integrated with traditional RL techniques such as the Deep Q-Network (DQN) algorithm on neuromorphic hardware [5], [6]. Although these previous methods demonstrated promising results, online, on-chip learning implementations are preferred, since on-chip learning both obviates the need to use high-power learning off chip and also performs learning in the processor being used to control

This work was funded by the NNSA Advanced Simulation and Computing Beyond Moore's Law project (GM) and the NNSA NA-22 program (ATS). GTM acknowledges funding from a GEM Fellowship funded by the National GEM Consortium for graduate funding.

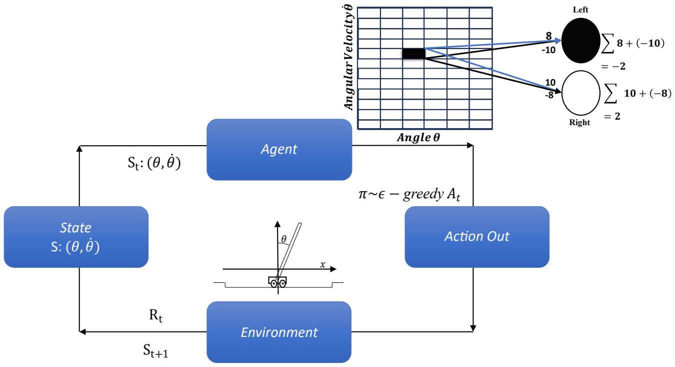

Fig. 1: Overview of spiking closed-loop Q-learning circuit. Here, the agent outputs actions, which are sent to the environment, OpenAI Gym, which determines the reward, and the control force on the cartpole determines a new state. The resulting reward and new state are then returned to the agent. Weight updates are performed with a greedy RL policy.

the agent. RL algorithms have been claimed to be biologically plausible [11]; but difficult to implement on neuromorphic processors due to several issues: (a) Continuous time and space interaction – the environment must be sampled after every action by a discrete state representation, therefore methods to update the proper state-action value are needed [1]; (b) Weight transport – errors from temporal computation of current and future Q-values need to be stored [1]; (c) System constraints – certain neuromorphic processors often have constraints on the compatible plasticity mechanisms that can be implemented [3].

II. METHODOLOGY

Q-Learning is a model-free, off-policy reinforcement learning algorithm that learns by finite Markov decision processes (finite MDP). Given any state, S_t, and action A_t, the agent learns by interacting with its environment (OpenAI-GYM) to learn the Q-matrix for the state-action values, which is updated to maximize future expected rewards. The agent uses the reward signal with an $\epsilon-$greedy policy to select actions from the Q-matrix. This process exemplifies a model-free architecture using off-policy Temporal Difference (TD) Learning. The TD control algorithm known as Q-Learning [1] is defined by

$$TD_{\text{error}} = \alpha \left[R_t + \gamma \max Q(S_{t+1}, a) - Q(S_t, A_t) \right] \quad (1)$$

$$Q_{\text{new}} = Q(S_t, A_t) + TD_{\text{error}}. \quad (2)$$

Q-learning and reinforcement learning face a challenge in balancing exploration and exploitation. Agents must explore the environment to gather rewards, but also exploit this knowledge to make optimal decisions and maximize rewards. Balancing exploration and exploitation is crucial to avoid suboptimal policies, as exploitation without exploration can miss out on better long-term strategies, while excessive exploration without exploitation can fail to capitalize on acquired knowledge. Several strategies have been proposed to balance exploration and exploitation in Q-learning. One common approach is ϵ-greedy policy [1], where the agent selects an action that has maximal estimated action value with probability ϵ of selecting a random action. The value of ϵ is typically reduced over time as the agent becomes more confident in its learned policy, allowing it to gradually shift from exploration to exploitation.

In our neuromorphic algorithm, $Q(S_t, A_t)$ is the Q-value that will be used in the weight update. The Q-value for the weight update is represented by the accumulation of u_t, the synaptic current, at time steps 2 and 5, Fig. 2. These values are stored as variables in neural registers and then used in the weight update; effectively representing $Q(S_t, A_t)$ and $\max Q(S_{t+1}, a)$. The synaptic current, u_t is calculated according to a standard leaky-integrate-and-fire (LIF) model

$$u_i(t) = \alpha u_i(t-1) + \sum_j w_{ij} s_j(t). \quad (3)$$

In the above equation, α is the current decay factor, w_{ij} are the synaptic weights, and s_j is a binary function representing a spike from neuron j. The membrane potential is calculated using

$$v_i(t) = \beta v_i(t-1) + u_i(t-1). \quad (4)$$

Here, β is the membrane potential decay factor. In the event the membrane potential exceeds the voltage threshold of the neuron, a spike is emitted and the potential resets to zero. The closed-loop representation of our system is shown in Fig. 1 and the learning parameters of our SNN for training are shown in Table I.

The network used for learning has only two layers: the pre-synaptic layer that represents the state S_t and the post-synaptic layer that represents A_t. Fig. 1 illustrates the argmax method used for closed-loop action selection, when a spike arrives at the pre-synaptic layer S_t, the synaptic weight $w(t)$ from (3) for the neuron is calculated by subtracting each weight (w_i, w_j) by its counterpart for the other action, for instance $w_{i,right} = q_{i,right} - q_{i,left}$. The neuron corresponding to the higher Q-value receives a positive potential, while the other receives a negative potential. This ensures that only the neuron with the highest Q-value reaches the firing threshold and emits a spike, effectively implementing the argmax function through spiking dynamics.

A. Cartpole Problem

The Q-learning algorithm computes the optimal action based on the state of the system to apply the correct force required to keep the pole upright. Since the cartpole system has a continuous state space, we discretize the pole's angle, θ, and angular velocity, $\dot{\theta}$. We represent the state space as a matrix, which is then unrolled into a flattened vector. The discretized parameters are encoded as sparse binary vectors and passed through the pre-synaptic

Hyperparameter	**Value**
discount factor, γ	0.999
initial learning rate, α_0	1.0
final learning rate, α_1	0.1
initial exploration, ϵ_0	1.0
final exploration, ϵ_1	0.1
decay rate for learning rate scheduler, k	0.01
network architecture, connections [in, out] of Q	[72,2]

TABLE I: Q-learning spike network hyper-parameters

layer, the dynamics of the cartpole system are described by

$$\ddot{\theta} = \frac{g\sin\theta + \cos\theta\left(\frac{-F-m_pl\dot{\theta}^2\sin\theta}{m_c+m_p}\right)}{l\left(\frac{4}{3}-\frac{m_p\cos^2\theta}{m_c+m_p}\right)} \tag{5}$$

$$\ddot{x} = \frac{F + m_pl\left(\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta\right)}{m_c+m_p}. \tag{6}$$

Here, F is the force applied to the pole, m_p is the mass of the pole, m_c is the mass of the cart, $\ddot{\theta}$ is the angular acceleration of the angle, $\ddot{x}$ is the linear acceleration, and g is the gravitational constant.

B. Training a Q-learning SNN

The neuromorphic circuit that implements Q-learning is depicted in Fig. 2. The figure depicts a network that learns to synaptically-encode the Q-matrix using SGSCs with bit-accurate LIF neurons. These neurons use the same dynamics as Loihi 2, but simulated with Lava. In traditional computing architectures the Q-learning algorithm has 32-bit weight precision for performing weight updates. Due to the hardware constraints of Loihi 2, our algorithm learns the Q-matrix with 8 bits of precision. Fig. 3 shows the weight update, which happens every 7 time steps due to periodic potentiation provided by the SGSC. Spike arrivals from the state and action neurons are stored in temporary memory relay neurons, r_{A_t}, r_{S_t}, and r_{S_t+1} which are used to perform a weight update. To perform weight update, y_2 the learning trace is used and sent to the learning engine.

C. Learning Rule

The learning rule implemented on the Loihi 2 learning engine is defined by the following equations

$$\Delta r = u_0(t) \cdot (y_2(t) - r(t)) \tag{7}$$

$$\Delta w = r(t) \cdot x_0(t) \cdot y_0(t). \tag{8}$$

In these equations, r is an intermediate weight update parameter, introduced due to constraints in weight update precision. The variable u_0 serves as the learning engine's indicator to apply weight updates at each learning epoch t. The variables $x_0(t) = S_{r_{S_t}}(t)$ and $y_0(t) = A_{r_{A_t}}(t)$ represent stored data derived from pre-synaptic and post-synaptic spikes, respectively. These values are held in memory neurons r_{S_t} and r_{A_t} (see Fig. 2). Equation (7) updates the intermediate update parameter, r, and (8) implements the local weight update rule.

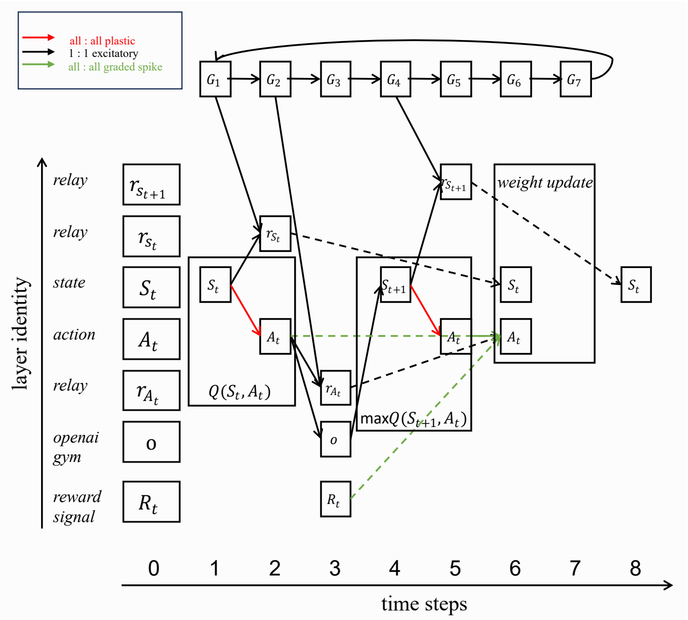

Fig. 2: Functional connectivity of the Q-learning circuit. Layer identity is indicated vertically (y-axis). Time is indicated horizontally (x-axis). The box in the upper left indicates connection type. The SGSC that controls (potentiates) neural information in the circuit is shown at the top (boxes labeled by G_i, with $i \in \{1, \ldots, 7\}$). Layers labeled with r indicate relay or short-term memory registers. o represents OpenAI Gym input. S_t indicates the state layer (a one-hot vector of $72 = 12 \times 6$ elements encoding a state with 12 angular velocities, 6 angles), A_t indicates the action layer of 2 neurons (indicating left and right actions). Dashed lines indicate delayed connections downstream. The subcircuits that compute (a) $Q-$value, (b) $\max Q$, and (c) the weight update. The $\max Q$ operation results from greedy action selection. Note that A_t and S_t are gated to pre- and post-synaptic sides of the plastic weights connecting the state and action layers such that the weight is updated with the Hebbian learning rule given in (8).

Timing for spike train transmission from the memory registers is coordinated by delayed synapses along with a gating pulse, ensuring proper synchronization during the learning process.

D. Learning Rate and Exploration

The success of the learning process relies on effectively balancing the agent's learning rate, α, and exploration rate, ϵ, to adapt over time without forgetting past experiences. Our learning rate scheduler dynamically adjusts these variables based on the rewards per episode and the total number of episodes experienced, ensuring a gradual and balanced progression from exploration to exploitation. The equations governing α are as follows

$$\beta = e^{-k \cdot E} \tag{9}$$

$$\alpha = \max(0.1, \min(1, 1 - \frac{\log_{10}(t+1)}{25})) \cdot \beta. \tag{10}$$

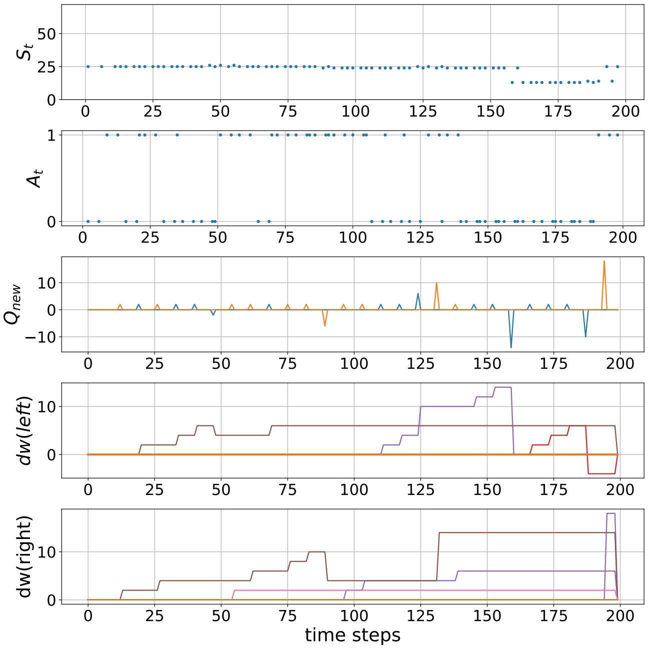

Fig. 3: Weight updates during a single episode. We plot S_t (72 neurons), A_t (2 neurons), Q_{new} (top three panels), and dw_{left} and dw_{right} (lower two panels). Given S_t and A_t the Q-learning circuit in Fig. 2 computes Q_{new}, which determines the updates dw_{left} and dw_{right}. Note that upticks in the (blue, orange) trace in Q_{new} prompt updates in dw_{left}, dw_{right}, respectively.

Here, β is a decay factor influenced by the number of episodes E, with k controlling the rate of decay. The learning rate α, modulated by β, determines how much weight the agent places on new experiences relative to prior knowledge. When α is high, the agent prioritizes incorporating new information to adapt to the environment, facilitating rapid learning. As α decreases over time, the agent increasingly relies on accumulated knowledge, stabilizing its decision-making process.

The exploration rate ϵ, is updated to match α during the training run. This linkage ensures that both learning and exploration are aligned: as the learning rate decreases, the agent reduces exploration and shifts its focus to exploiting learned strategies. The gradual reduction of α and ϵ ensures the agent transitions effectively from exploration to exploitation, refining its actions based on learned policies without oversaturating with new information.

This dynamic scheduler is critical for achieving stable and efficient learning in environments that require both adaptability and retention of past experiences. By coupling the learning and exploration rates, our framework effectively balances the trade-off between exploration of the environment and exploitation of learned strategies, contributing to the robustness and efficiency of the overall learning algorithm.

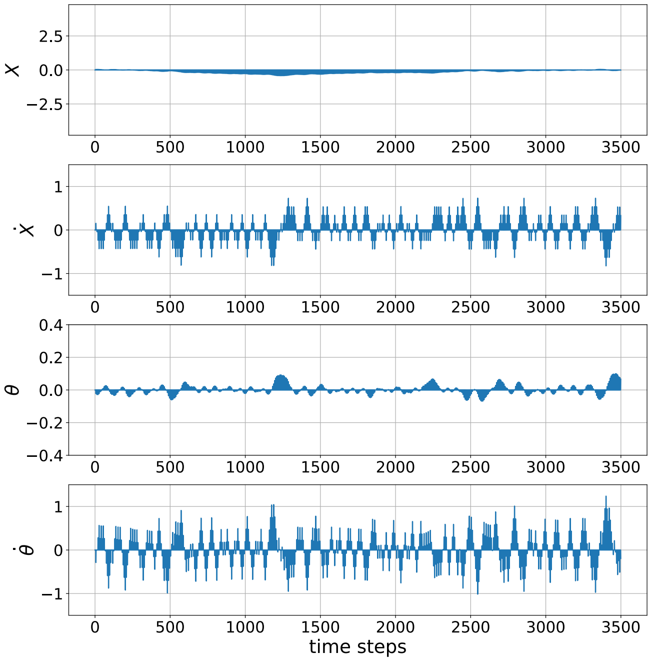

Fig. 4: Evolution of x, $\dot{x}$, θ, and $\dot{\theta}$ as a function of time for a single test episode. These results demonstrate that the cartpole maintains its location with little deviation over the duration of the test. The time derivatives of x and θ oscillate such that the cartpole remains upright and the angle, θ, relative to 0 (exactly upright) also oscillates as the cartpole remains stable.

E. Gating Chain

Gating neurons are organized in a ring structure (Fig. 2), analogous to the voltage-controlled oscillator (VCO) in a CPU that generates clock signals. This configuration enables the gating neurons to autonomously manage the propagation of information within the neuronal circuit, mimicking the timing benefits of a CPU without relying on traditional sequential processor control. The gating mechanism functions as a logical AND gate or coincidence detector, ensuring that spikes in the gating chain align with spikes in target neuronal layers (registers) to facilitate efficient information routing. This approach provides precise timing for the execution of computational steps in the circuit. In our implementation, the gating chain comprises 7 neurons, corresponding to 7 algorithmic time steps required for the learning process. The gating neurons are connected to the relay neurons by positive weight connections. The relay neurons have a voltage threshold greater than the synapses that propagate incoming spikes from the gating neurons. The relay neurons will only spike when gating neurons G_1, G_2, and G_4 fire a spike that arrives at the same time as the spikes from S_t and A_t as shown in Fig. 3.

III. RESULTS

This manuscript presents the design and implementation of a self-contained, spike-based neural circuit capable of solving

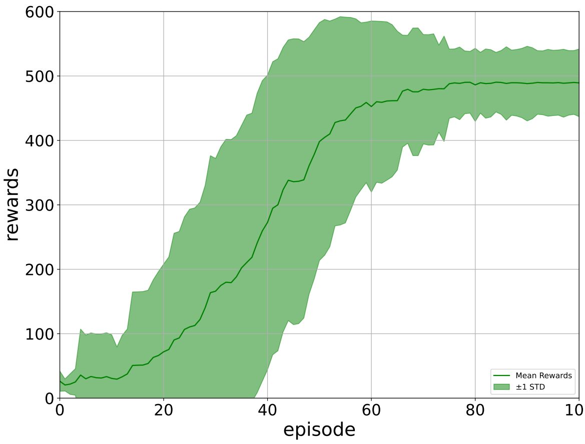

Fig. 5: Evolution of the reward over 100 episodes. Rewards, R_t, from the training of 50 agents are shown. The agent's mean is depicted with dark green and the standard deviation is depicted as the lighter green region surrounding the mean. Note that, after 60 episodes, the agents are trained with a lower bound on the reward of approximately 350, and an average reward of approximately 475.

the cartpole problem. This section outlines the evaluation process of our learning algorithm and provides performance results from 50 independent random simulations, each running for at least 100 episodes.

Figure 4 illustrates the evolution of the parameters (x, $\dot{x}$, θ, $\dot{\theta}$) of the cartpole system during a single episode, including the pole angle, angular velocity, cart position along the X-axis, and linear acceleration over time. Notably, the pole angle remains consistently near zero degrees, demonstrating the algorithm's ability to effectively maintain the pole's balance throughout the simulation. To evaluate training, we measured the standard deviation of accumulated rewards across 50 random simulations. As anticipated, our algorithm consistently improved its performance reaching an average reward of approximately 475 after 60 episodes, as shown in Fig. 5.

With this reward level, the cartpole was consistently balanced over the length of an episode (Fig. 5). In our tests of the cartpole balancing agent, the system was able to successfully balance the cartpole for initial conditions previously unseen in the training data (see Fig. 6). These results validate the proposed learning algorithm's capacity to maintain stability and solve the cartpole problem within a spiking neural network.

IV. Discussion

This work demonstrates that spiking neural networks with local learning can successfully implement on-chip reinforcement learning (RL) algorithms within modular neuronal circuits, providing a foundation for online learning that can be translated to neuromorphic hardware. By leveraging synfire-gated synfire chains (SGSCs), we showcased the ability to control spike propagation and facilitate real-time learning, highlighting the potential for spiking networks to perform sophisticated computational tasks.

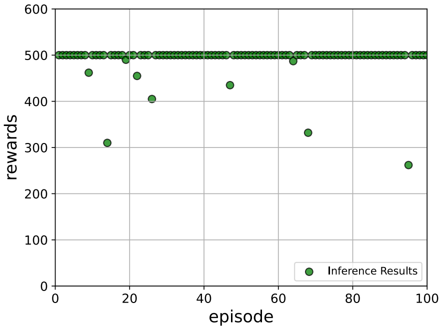

Fig. 6: This figure presents the results of inference performed after training a high-performing agent. Each episode represents an independent test conducted over 3,500 time steps. Notably, the agent demonstrates robust performance, successfully balancing the cartpole in all episodes. The mean score across these 100 episodes is 491.38, highlighting the agent's consistency and effectiveness.

Although on-chip learning was only simulated in this study, our results establish a proof of principle for online learning in spiking networks, paving the way for future neuromorphic implementations. Specifically, future work will focus on fully implementing Q-learning directly on neuromorphic hardware, where the learning of synaptic weights will be achieved using intrinsic neuronal mechanisms inherent to the hardware. This transition will enable a more integrated and efficient approach to solving control problems on energy-efficient platforms.

By bridging the gap between algorithmic design and neuromorphic implementation, this study underscores the viability of SGSCs as a mechanism for implementing online learning in spiking neural networks. Our framework provides a pathway for advancing neuromorphic RL, ultimately contributing to the development of more adaptive and energy efficient AI systems.

Acknowledgment

We thank the Intel Neuromorphic Research Community for access to Lava and help with their software framework. The LANL designation for this document is LA-UR-24-32562.

References

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[2] Eva García-Martín, Crefeda Faviola Rodrigues, Graham Riley, and Håkan Grahn. Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, 134:75–88, 2019.

[3] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[4] Alpha Renner, Forrest Sheldon, Anatoly Zlotnik, Louis Tao, and Andrew Sornborger. The backpropagation algorithm implemented on spiking neuromorphic hardware. *Nature Communications*, 15(1):9691, 2024.

[5] Mahmoud Akl, Yulia Sandamirskaya, Deniz Ergene, Florian Walter, and Alois Knoll. Fine-tuning deep reinforcement learning policies with r-stdp for domain adaptation. In *Proceedings of the International Conference on Neuromorphic Systems 2022*, ICONS '22, New York, NY, USA, 2022. Association for Computing Machinery.

[6] Mahmoud Akl, Yulia Sandamirskaya, Florian Walter, and Alois Knoll. Porting deep spiking q-networks to neuromorphic chip loihi. In *International Conference on Neuromorphic Systems 2021*, ICONS 2021, New York, NY, USA, 2021. Association for Computing Machinery.

[7] Andrew T Sornborger, Zhuo Wang, and Louis Tao. A mechanism for graded, dynamically routable current propagation in pulse-gated synfire chains and implications for information coding. *Journal of computational neuroscience*, 39:181–195, 2015.

[8] Zhuo Wang, Andrew T Sornborger, and Louis Tao. Graded, dynamically routable information processing with synfire-gated synfire chains. *PLoS computational biology*, 12(6):e1004979, 2016.

[9] G Brockman. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[10] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[11] Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in Neural Circuits*, 9, 2016.