A Novel Reward-driven Metropolis-Hastings Sampling Method for Dynamic Agent Control via Spiking Neural Networks

Ali Safa[1], Farida Mohsen[1], Ali Al-Zawqari[2]
[1]*College of Science and Engineering, Hamad Bin Khalifa University, Doha, Qatar*
[2] *Department of Fundamental Electricity and Instrumentation, Vrije Universiteit Brussel, Brussels, Belgium*
asafa@hbku.edu.qa, fmohsen@hbku.edu.qa, aalzawqa@vub.be

***Abstract*—Spiking Neural Networks (SNNs) promise low-power, real-time control but their training can be challenging for reinforcement learning (RL) tasks, due to the non-differentiability of spikes and the hardware non-idealities of emerging analog and mixed-signal neuromorphic processors. Whithin this context, we present a gradient-free framework for training SNN policies with Metropolis-Hastings (MH) sampling, using episode returns as a reward-driven pseudo-likelihoods to propose, accept or reject network parameter updates. In evaluations on standard RL benchmarks (AcroBot and CartPole), our single-layer SNN policies outperform deep Q-learning (DQL) baselines while using fewer neurons and training episodes: Acrobot reaches -90 vs. -140 (for DQL), and CartPole achieves the maximum score of 500 vs. 280, plateauing within ≈ 50 episodes. These results highlight the effectiveness of MH-trained SNNs for control and their suitability for neuromorphic deployment.**

***Index Terms*—Spiking Neural Networks, Metropolis-Hastings sampling, Dynamical Agent, Control, Reinforcement Learning.**

I. Introduction

Reinforcement learning (RL) has achieved strong results in control using deep Q-learning (DQN) and proximal policy optimization (PPO) [1]–[3]. However, these methods require substantial computation and rely on gradient backpropagation, which limits the deployment on ultra-low power platforms and mixed-signal neuromorphic hardware [4].

Spiking Neural Networks (SNNs) offer event-driven efficiency, temporal processing, and biological plausibility, making them attractive for low-power real-time control [5]–[7]. However, training SNNs for RL remains challenging: spike events are nondifferentiable, so surrogate gradient methods are required, and these are computationally intensive and less robust on analog substrates with process–voltage–temperature (PVT) variations [8]. This motivates gradient-free optimization that is robust to hardware non-idealities.

We propose a Metropolis–Hastings (MH) reward–driven procedure that trains SNN policies without gradients by accepting parameter proposals in proportion to episode return [9], [10]. The framework naturally supports chip-in-the-loop optimization and robustness to hardware noise/variability [11]. We extend MH sampling to RL settings, specifically focusing on dynamical-agent control (Fig. 1). The key contributions of this paper are:

1) Introducing MH sampling for training SNNs in RL-based dynamical-agent control tasks.
2) Demonstrating that MH-trained SNNs outperform traditional DQL baselines on standard RL benchmarks (AcroBot and CartPole).
3) Achieving stronger generalization and performance with simpler network architectures than gradient-based methods.

A. Safa supervised the project as Principal Investigator, contributed to the technical developments and to the writing of the manuscript. F. Mohsen contributed to the technical analysis and to the writing of the manuscript. A. Al-Zawqari contributed to the writing of the manuscript and provided technical feedback.

Fig. 1: ***Environments used in our experiments for the control of dynamical agents.*** *a)* `AcroBot`*, b)* `Cart-Pole`

II. Background Theory

A. Metropolis-Hastings sampling

In this work, we use the Metropolis-Hastings sampling algorithm [10] as a Bayesian training approach to learn the SNN weights. Metropolis-Hastings provides a computationally tractable way for estimating the posterior distribution of the SNN weights $P(W|D)$ given the data D following Bayes' rule [10]:

$$P(W|D) = \frac{L(D|W)P(W)}{P(D)} \tag{1}$$

where, in the context of this paper, W denotes the SNN weights, $L(D|W)$ is the likelihood probability distribution of the data given the SNN weights, $P(W)$ is the prior distribution, $P(D)$ is the data distribution and $P(W|D)$ the posterior distribution of W.

It is commonly known in the Bayesian inference literature that the *explicit* use of the Bayes rule (1) typically comes with a high computational complexity the more the dimensions of the weight tensor W grows [**?**]. For this reason, we make use of the Metropolis-Hastings sampling algorithm for *implicitly* estimating the posterior density over the weights $P(W|D)$. Metropolis-Hastings estimates $P(W|D)$ by providing a sequence of weight tensor samples $W \sim P(W|D)$ which provably correspond to true samples drawn from the distribution $P(W|D)$ [10]. Remarkably, these samples are obtained without an exact knowledge of the posterior distribution $P(W|D)$.

Fig. 2 provides a conceptual illustration of the Metropolis-Hastings sampling algorithm. First, the sampler is initialized with a random weight tensor W_1 at sampling step $n = 1$. Then, a symmetric distribution Q (typically a Gaussian) is used to obtain a proposal sample $W^{'} \sim Q(W^{'}|W_{n-1})$ using W_{n-1} as the mean value of Q. Second, the likelihoods of the data D

given the newly-proposed weight $W^{'}$ and the previous weight W_{n-1} are computed as $L(D|W^{'}), L(D|W_{n-1})$ (where L is an arbitrary-chosen likelihood function suitable for the task to be solved). Third, the following ratio p is computed:

$$p = \frac{L(D|W^{'}P(W^{'})}{L(D|W_{n-1})P(W_{n-1})} \tag{2}$$

where $P(W^{'})$ and $P(W_{n-1})$ respectively denote the prior probability of $W^{'}$ and W_{n-1}. Finally, the newly-proposed sample $W^{'}$ is accepted as a true sample drawn from $P(W|D)$ if and only if the outcome of a binary Bernoulli trial with probability $\min(p, 1)$ is successful.

B. Spiking Neural Networks

We control dynamical agents with SNNs, whose event-driven computation is well suited to low-power, real-time control [12], [13]. We use leaky integrate-and-fire (LIF) neurons: given input current $J[m] = \bar{\mathbf{w}}^{\top}\bar{\mathbf{s}}[m]$, the membrane potential updates as

$$V[m] \leftarrow \alpha V[m-1] + (1-\alpha)J[m], \tag{3}$$

and when $V[m] \geq \mu$ the neuron emits a spike (δ_{out}=1) and resets $V[m] \leftarrow 0$. Using LIF units instead of analog activations (e.g., ReLU) enables spike-based communication and temporal dynamics closer to biological processing [13].

III. Methods

In this paper, we propose to leverage MH sampling for learning to control dynamical agents within a RL-like setting. In such RL setting, the environment in which the dynamical agent operates returns rewards r_j for each time step j during the execution of an agent control episode, with a maximum number of time step per episode $j = 1, ..., T_{max}$ during which the agent is expected to attain its goal. It is assumed that the agent control is done through an arbitrary SNN architecture and we note by W the tensor containing all of the learnable SNN parameters (e.g., weights, neural membrane decay constant α_{decay} and spiking threshold μ). Hence, the optimization objective of the SNN learning process can be written as the maximization of the accumulated reward:

$$W_{best} = \arg\max_{W} \sum_{j=1}^{T_{max}} r_j(W) \tag{4}$$

Here, we note a similarity between the SNN learning objective (4) and the objective of MH sampling, which is to provide SNN parameter samples that will maximize a certain likelihood $L(D|W)$ in (1). Hence, by defining:

$$L \equiv R(W) = \sum_{j=1}^{T_{max}} r_j(W) \tag{5}$$

where $R(W)$ denotes the total accumulated reward over an episode, it is possible to use the pseudo-likelihood measure L in (5) in place of an exact likelihood function within the MH sampling procedure. This is possible thanks to the fact that MH sampling makes use of the *ratio* between the current and past likelihood functions in (2), which makes it possible to drop the requirement for an exact knowledge of the normalization terms defining the *absolute* likelihood function.

Following these observations, our proposed reward-driven MH algorithm for SNN training is provided in Algorithm 1. The algorithm follows the MH sampling procedure illustrated in Fig. 2 with the crucial addition of the accumulate rewards $R(W)$ and $R(W_{n-1})$ as likelihood values in lines 4-5-6 of Algorithm 1.

Algorithm 1 Proposed Reward-driven Metropolis-Hastings method for SNN training

Require: Gym environment (e.g., `AcroBot` and `Cart-Pole`), initial weights W_0

1: Init. $R_{best} = -\infty$, $W_{best} = W_0$
2: **for** $n = 1$ to N_{iter} **do**
3: Sample proposal $W' \sim Q(W'|W_{n-1})$
4: Run the gym agent control environment with the SNN with weights W' and compute the total accumulated reward $R(W')$.
5: Run the gym agent control environment with the SNN with previous weights W_{n-1} and compute the total accumulated reward $R(W_{n-1})$.
6: Assign likelihoods: $L_1 \leftarrow R(W')$, $L_2 \leftarrow R(W_{n-1})$
7: Compute priors: $P_1 \leftarrow P(W')$, $P_2 \leftarrow P(W_{n-1})$
8: Compute acceptance ratio: $p = \frac{L_1 \cdot P_1}{L_2 \cdot P_2}$
9: Sample $u \sim \text{Uniform}(0, 1)$
10: **if** $u < \min(p, 1)$ **then**
11: Accept proposal: $W_n \leftarrow W'$
12: **else**
13: Reject proposal: $W_n \leftarrow W_{n-1}$
14: **end if**
15: **if** $R(W') > R_{best}$ **then**
16: $R_{best} \leftarrow R(W')$
17: $W_{best} \leftarrow W_n$
18: **end if**
19: **end for**
20: Return W_{best}

IV. Experiments

We evaluate the proposed method on two classic control environments from the OpenAI Gym suite [14]: `AcroBot` and `Cart-Pole` (Fig. 1). As a baseline, we compare against DQL [2].

We use a public DQL implementation[1] with Adam at default settings, standard experience replay, and an ϵ-greedy policy [15]. We found the default hyperparameters sufficient in terms of convergence speed and return; additional tuning did not improve results.

As an SNN architecture, we use the 1-layer topology shown in Fig. 3. The SNN has two sets of weights: an input weight matrix W_{in} mapping the agent's state vector $\bar{s}$ to the output LIF neurons, and a lateral inhibition and recurrence weight matrix $W_{lateral}$ feeding back the output spike vector $\bar{u}$ to the output neurons. The decay constant α_{decay} and threshold μ of the output LIF neurons are also tuned by the proposed MH training procedure. Naturally, the input and output dimensions depend on the specifications of each environment to be solved (`AcroBot` having 6 states, and `Cart-Pole` having 4 states. During the training procedure, the MH algorithm provides samples of vectors containing an estimate of the network parameters $\{W_{in}, W_{lateral}, \alpha_{decay}, \mu\}$ with the goal of maximizing the accumulated reward returned by the environment (see Algorithm 1).

A. AcroBot Environment

The `AcroBot` task is a two-link underactuated pendulum that must swing up so the tip reaches a goal line above the

[1] `https://github.com/johnnycode8/gym_solutions`

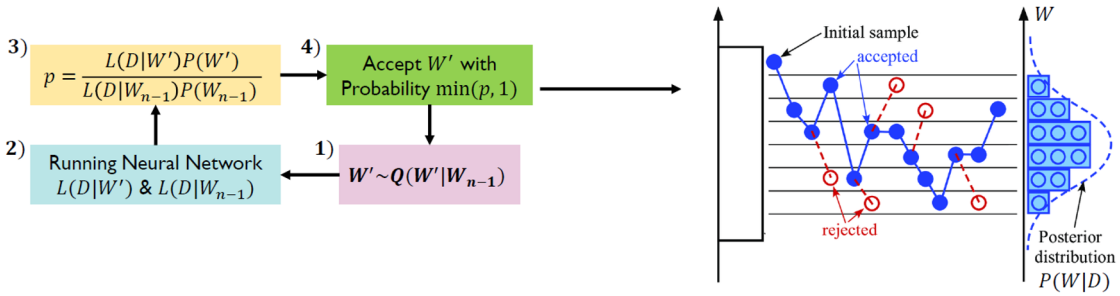

Fig. 2: ***Conceptual illustration of the Metropolis-Hastings algorithm.***

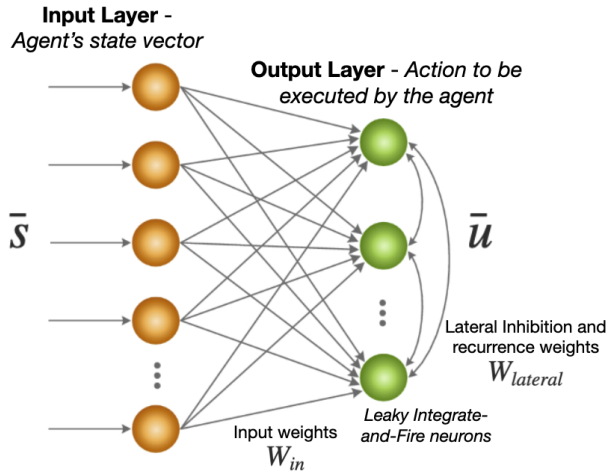

Fig. 3: ***SNN architecture.*** *We use a 1-layer SNN topology with input weight matrix W_{in} and lateral recurrence weight matrix $W_{lateral}$. As input to the network, we feed the agent's state vector (with the input dimension depending on the environment to be solved). The output is then used to select the action to be executed by the agent (each output neuron corresponding to a possible action).*

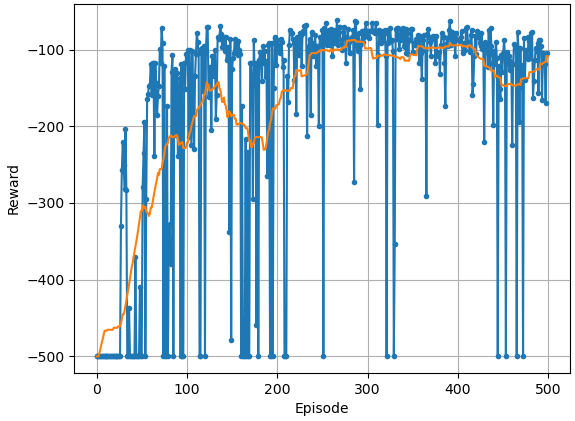

Fig. 5: Evolution of accumulated reward during MH sampling on the Acrobot environment. The orange curve is a 50-episode moving average of the blue reward trace.

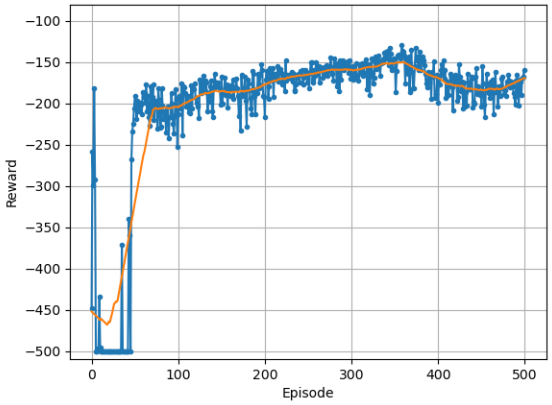

Fig. 6: **AcroBot—DQL training return.** Episode return vs. iteration; orange is a moving average (window 50). DQL plateaus near ~ -150 (cf. SNN–MH ~ -100 in Fig. 5).

pivot (Fig. 1a). The agent observes a 6-D state related to the angular position of the pendulum joints and their angular velocities.

Figure 5 shows the training return under MH. After an initial transient of $\sim$160 episodes, the return increases and plateaus near -100, which corresponds to solving the task in roughly 100 time steps per episode (the environment returns -1 per step until termination).

In addition, Fig. 6 shows the results obtained using the DQL baseline (using a similar DNN topology to the SNN architecture shown in Fig. 3, with the LIF neurons replaced with ReLUs). It can be seen that the DQL setup achieves a reward plateau with a significantly smaller accumulated reward of at most ~ -150, while our proposed SNN-MH setup achieves an accumulated reward of ~ -100 (see Table I).

B. Cart-Pole Environment

The `Cart-Pole` environment consists of a cart that can move horizontally with an inverted pendulum mounted on its top (see Fig. 1 b). The goal is to control the cart in order to keep the inverted pendulum vertical and not let it fall due to gravity. The agent has 4 observable states related to the position and velocity of the cart, as well as the angle and angular velocity of the inverted pendulum.

Fig. 7 shows the evolution of the accumulated reward during the MH training procedure on the `Cart-Pole` environment. It can be seen that the accumulated reward grows and reaches within 50 episodes a reward plateau around 500 (which is the maximum reward attainable in the case of the `Cart-Pole` environment).

In addition, Fig. 8 shows the results obtained using the DQL baseline. Remarkably, DQL was not able to solve the `Cart-Pole` environment using a similar DNN topology as used by our SNN (i.e., a 1-layer network). This is why we experimented with an increasing amount of hidden layers in the DNN (using 16 neurons in each hidden layer). Fig. 8 shows that the maximum accumulated reward grows as we increase the number of hidden layers. At the 3 hidden layers, the DQL

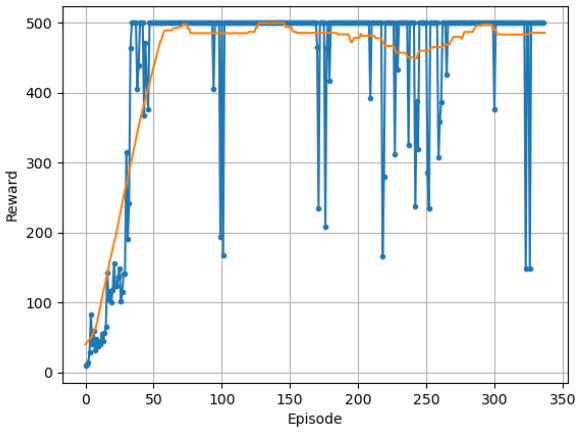

Fig. 7: ***Evolution of the accumulated reward along the MH sampling episodes.*** *Using the* `Cart-Pole` *environment. The orange line was obtained by applying a moving average filter of width* 50 *on the reward curve in blue.*

setup is finally able to solve the `Cart-Pole` environment by reaching an accumulated reward of 500. This is in striking contrast with our proposed SNN-MH setup which is able to solve `Cart-Pole` by reaching a reward plateau of 500 (see Fig. 7) while using the 1-layer SNN architecture described in Fig. 3. The comparison between the SNN-MH setup and the DQL setup is also reported in Table I.

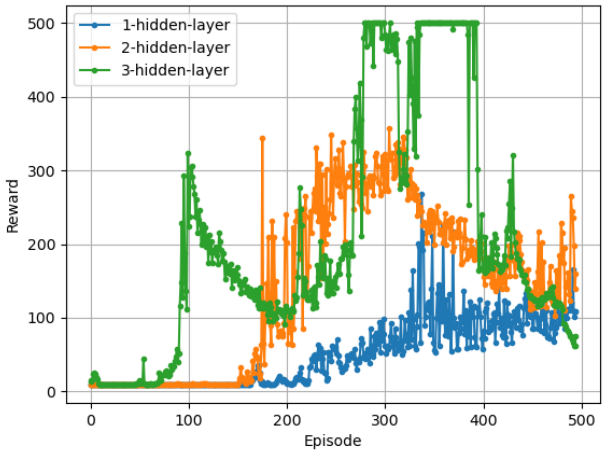

Fig. 8: **DQL**: evolution of accumulated reward in the CartPole environment. In contrast to our proposed SNN-MH setup, using a one- or even two-hidden-layer network in the DQL case is insufficient to solve the environment (i.e., reach the maximum reward of 500).

	SNN-MH	**DQL**
`AcroBot`	**−90**	−140
`Cart-Pole`	**500**	280

TABLE I: ***Maximum accumulated reward*** *for both the SNN-MH and the DQL with equivalent 1-hidden-layer architecture (rewards are rounded to the nearest decade).*

C. Discussion on the results

Our findings show the effectiveness of training SNNs with MH sampling in controlling dynamic agents within RL environments. Firstly, MH sampling improves generalization compared to the gradient-based optimization used in DQL by exploring the Bayesian posterior and thus better avoiding local optima [10], [16]. Concurrently, the event-driven, sparse activations of SNNs act as an inductive bias, regularizing the policy network and simplifying decision boundaries. This synergy underlies our method's fast convergence and high reward performance on both AcroBot and CartPole. Table I clearly demonstrates the usefulness of our SNN-MH setup, where the reward-driven MH sampling method proposed in this work systematically reaches a higher accumulated reward than DQL.

Finally, the *gradient-free* nature of the MH sampling method used in this work confers robustness to hardware non-idealities during chip-in-the-loop training [11]. Its MH stochastic acceptance mechanism tolerates analog noise, voltage fluctuations, and timing jitter, making it particularly suitable for neuromorphic chip-in-the-loop implementations.

V. Conclusion

We introduced a reward-driven, gradient-free procedure that trains SNN control policies via MH sampling, using episode return in place of a likelihood. On `AcroBot` and `Cart-Pole`, a single-layer SNN trained with our method achieved higher returns and faster convergence than DQL baselines, while avoiding backpropagation and remaining compatible with neuromorphic chip-in-the-loop training. Future work includes adaptive proposal distributions and scaling to more complex tasks and hardware deployments.

References

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

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

[4] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro*, 38(1):82–99, 2018.

[5] Samanwoy Ghosh-Dastidar and Hojjat Adeli. Spiking neural networks. *International journal of neural systems*, 19(04):295–308, 2009.

[6] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

[7] Katsumi Naya, Kyo Kutsuzawa, Dai Owaki, and Mitsuhiro Hayashibe. Spiking neural network discovers energy-efficient hexapod motion in deep reinforcement learning. *Ieee Access*, 9:150345–150354, 2021.

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

[9] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335, 1995.

[10] C.P. Robert. The metropolis–hastings algorithm. In N. Balakrishnan, T. Colton, B. Everitt, W. Piegorsch, F. Ruggeri, and J.L. Teugels, editors, *Wiley StatsRef: Statistics Reference Online*. Wiley, 2015.

[11] A. Safa et al. Towards chip-in-the-loop spiking neural network training via metropolis-hastings sampling. *Proc. Neuro Inspired Comput. Elements Conf. (NICE)*, pages 1–5, 2024.

[12] A. Safa, I. Ocket, A. Bourdoux, H. Sahli, F. Catthoor, and G. G. E. Gielen. Stdp-driven development of attention-based people detection in spiking neural networks. *IEEE Transactions on Cognitive and Developmental Systems*, 16(1):380–387, February 2024.

[13] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol. A 0.086-mm² 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos. *IEEE Transactions on Biomedical Circuits and Systems*, 13(1):145–158, February 2019.

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

[15] Aske Plaat. *Deep Reinforcement Learning*. Springer Nature Singapore, 2022.

[16] L. V. Jospin, H. Laga, F. Boussaid, W. Buntine, and M. Bennamoun. Hands-on bayesian neural networks—a tutorial for deep learning users. *IEEE Computational Intelligence Magazine*, 17(2):29–48, May 2022.