Spiking Variational Policy Gradient for Brain Inspired Reinforcement Learning

Zhile Yang, Shangqi Guo, Ying Fang, Zhaofei Yu, Jian K. Liu

Abstract—Recent studies in reinforcement learning have explored brain-inspired function approximators and learning algorithms to simulate brain intelligence and adapt to neuromorphic hardware. Among these approaches, reward-modulated spike-timing-dependent plasticity (R-STDP) is biologically plausible and energy-efficient, but suffers from a gap between its local learning rules and the global learning objectives, which limits its performance and applicability. In this paper, we design a recurrent winner-take-all network and propose the spiking variational policy gradient (SVPG), a new R-STDP learning method derived theoretically from the global policy gradient. Specifically, the policy inference is derived from an energy-based policy function using mean-field inference, and the policy optimization is based on a last-step approximation of the global policy gradient. These fill the gap between the local learning rules and the global target. In experiments including a challenging ViZDoom vision-based navigation task and two realistic robot control tasks, SVPG successfully solves all the tasks. In addition, SVPG exhibits better inherent robustness to various kinds of input, network parameters, and environmental perturbations than compared methods.

Index Terms—Spiking neural networks, reinforcement learning, reward-modulated spike-timing-dependent plasticity, winner-take-all circuit, variational policy gradient.

1 Introduction

Reinforcement learning (RL) based on artificial neural networks (ANNs) has gained success in many scenarios [1], [2], [3], [4]. However, the adaptability and robustness of the current models are still unsatisfactory compared to human intelligence. Although many studies have made good progress in improving these performances, they generally require specific designs, such as task augmentation [5], [6], [7], additional policy modules [8], [9], or task-specific policy structures [2]. A recent branch of RL studies focuses on biologically plausible learning systems that simulate biological neurons and network structures of human brains. It has been found that these systems can inherently improve the performance of artificial agents [10], [11], [12] and the energy efficiency of the learning system [13], thus reducing the need of extra training costs and task-specific designs. This branch of studies also contributes to the analysis of experimentally recorded neuronal signals, providing information to explain the capabilities of the brain in the field of computational neuroscience [14].

One of the mainstream biologically plausible models is spiking neural networks (SNNs). SNNs differ from conventional ANNs primarily in their spiking neurons, which model the activities as discrete spikes, making the gradient on network parameters unavailable. SNN RL methods can be categorized into three types: ANN-to-SNN (ANN2SNN) [15], surrogate gradient function [16], [17], and reward-modulated spike-timing-dependent plasticity (R-STDP) [18], [19]. The first two types require backpropagation of gradients during training and are not considered biologically plausible [20], [21], [22]. In contrast, R-STDP trains with local learning rules and thus are considered biologically plausible [22], [23], [24]. In addition, R-STDP is also more preferable for implementation by neuromorphic hardware [25]. Recent R-STDP studies investigated different forms of R-STDP, considering types of neuronal interactions, modulation strengths, and modulation timings [18], and have seen improvements in task performance and energy efficiency [10], [11], [26]. However, their approaches require task-specific designs, and it is also not clear whether and how the local R-STDP rules optimize global RL objectives with theoretical guarantees. Therefore, there exists a fundamental gap between the local learning rules and the overall RL target.

To bridge the gap, we adopt the variational inference method to transform the global learning target into local learning rules. This idea has been used in previous works on pattern generation and classification [27], [28], [29], but not in the RL community. To our knowledge, this paper is the first to investigate variational inference for R-STDP. We design an *RWTA network*, a spiking neural network that consists of recurrently connected winner-take-all (WTA) circuits [28], [30]. We show that the fixed point of the RWTA network is equal to an energy-based RL policy distribution and that R-STDP can be specified for the RWTA network to serve as an approximation of the policy gradient in the REINFORCE [31] algorithm. These establish the theoretical

- *Zhile Yang is with School of Computer Science, University of Leeds, Leeds, UK.*
  E-mail: sczy@leeds.ac.uk
- *Shangqi Guo is with Department of Precision Instrument, Department of Automation, Tsinghua University, Beijing, China.*
  E-mail: shangqi_guo@mail.tsinghua.edu.cn
- *Ying Fang is with College of Computer and Cyber Security, Fujian Normal University, Fujian, China.*
  E-mail: fy20@fjnu.edu.cn
- *Zhaofei Yu is with School of Artificial Intelligence, Peking University, Beijing, China.*
  E-mail: yuzf12@pku.edu.cn
- *Jian K. Liu is with School of Computer Science, University of Leeds, Leeds, and School of Computer Science and Centre for Human Brain Health, University of Birmingham, Birmingham, UK.*
  E-mail: j.liu.22@bham.ac.uk

equivalence between the R-STDP learning rules and the RL objective. The resulting algorithm is named the spiking variational policy gradient (SVPG).

We evaluate SVPG over five typical RL tasks including reward-based MNIST classification [32], Gym Inverted-Pendulum [33], ViZDoom HealthGathering [34], AI2THOR robot navigation [35], [36], and robot arm manipulation [37]. Among them, the ViZDoom task, as a 3D first-person video game, is the most challenging as it involves image input and long decision sequences. The robot tasks use near photo-realistic scenes [35], [37] and randomized starting/target positions, which examines the method's applicability to real-world tasks. To extend the capability of the SVPG algorithm, we enhance its base RL algorithm from REINFORCE to the more widely-used and efficient PPO-clip [38] algorithm and transplanted the Adam optimizer [39] and the RMSprop optimizer [40]. Empirical results show that SVPG can solve all five tasks, and outperforms three representative methods including ANN, ANN2SNN, and surrogate gradient-based backpropagation in terms of optimization speed. Furthermore, SVPG exhibits inherent robustness to input noise [15], network parameter noise [41], and environmental variation [42].

Note that a part of this work has been presented at the BMVC 2022 [43]. This article mainly incorporates these extensions: 1) the SVPG algorithm is extended from REINFORCE to more RL base algorithms, including PPO-clip, 2) three more challenging tasks are included in the experiments, and 3) more visualizations and properties of the network are investigated.

We summarize the main contributions of this work as follows:

- We propose SVPG, a biologically plausible learning method for a RWTA network. SVPG establishes a connection between the R-STDP framework and RL policy gradient algorithms.
- We experimentally demonstrate that SVPG is capable of solving a series of RL tasks, including two vision-based robot navigation tasks in realistic scenarios.
- We experimentally show that SVPG produces effective policies with inherent robustness to several types of perturbations.

2 Preliminary

2.1 Spike Response Model and Winner-Take-All Circuit

We adopt the spike response model (SRM) [44] for the neuron model. SRM is a stochastic variant of the leaky integrate-and-fire model [45], and has been widely used in RL studies [44], [45], [46], [47]. We also consider WTA circuits as base components in the network. Figure 1 illustrates SRM neurons in a WTA circuit.

In SRM, the states of neurons evolve at discrete spike time steps. At each spike time step l, each neuron evolves its membrane potential $u(l)$ and fires a spike at random according to its firing probability $\rho(l)$. All spikes are considered to be binary, i.e., 1 for firing and 0 for resting. The firing probability $\rho(l)$ depends exponentially on the membrane potential [44], as defined in the following equation:

$$\rho(l) = \exp\{u(l) - I(l)\}, \tag{1}$$

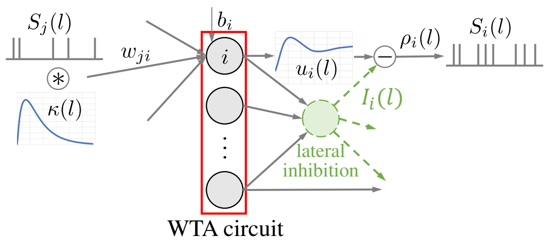


Fig. 1. The structure of a WTA circuit.

where $u(l)$ is the membrane potential and $I(l)$ is an inhibitory term produced by a lateral inhibition neuron in the WTA circuit. The inhibitory term is assumed to ensure that the firing probabilities within the WTA circuit sum up to 1, resulting in a low firing rate of the whole network and low energy consumption. It is also assumed that one and only one neuron in the WTA circuit fires at each time step. The membrane potential $u(l)$ is determined by the spike train S from connected neurons [48]:

$$u(l) = \sum_{j \in N(\cdot)} w_j \int_0^\infty \kappa(y) S_j(l-y) \mathrm{d}y + b, \tag{2}$$

where $N(\cdot)$ denotes the set of presynaptic neurons of the considered neuron, w_j is the synapse weight, κ is the excitatory postsynaptic potential, S_j is the spike train from neuron j, and b is the intrinsic excitability of the neuron. For κ, we consider the double exponential and the rectangle function [48], which are respectively:

$$\kappa(l) = \kappa_0[\exp(-l/\tau_1) - \exp(-l/\tau_2)]\varepsilon(l), \tag{3}$$

$$\kappa(l) = \kappa_0[\varepsilon(l) - \varepsilon(l-\tau_3)], \tag{4}$$

where κ_0 is the overall amplitude, τ_1, τ_2, and τ_3 are hyperparameters, and $\varepsilon(l)$ is the step function.

2.2 Reward-Modulated STDP

R-STDP is a framework of learning rules for synapse weights given the local firing activities and an external modulatory signal. It is supported by neuroscience findings about the relationship between dopamine, acetylcholine, and synapse plasticity [49], [50], and has been used in various biologically plausible RL studies [45], [46], [51]. We use w_{ij} to denote the weight of synapse between presynaptic neuron i and postsynaptic neuron j. R-STDP takes the form of $\Delta w_{ij} = R(l) \cdot \mathrm{STDP}(l)$, where $R(l)$ is the external reward signal and $\mathrm{STDP}(l)$ is a coefficient determined by the STDP learning rule in the following form:

$$\begin{aligned} \mathrm{STDP}(l) = S_j(l)\big[W_{\text{pre}} + \int_0^\infty A_+ W_+(y) S_i(l-y)\mathrm{d}y\big] \\ + S_i(l)\big[W_{\text{post}} + \int_0^\infty A_- W_-(y) S_j(l-y)\mathrm{d}y\big], \end{aligned} \tag{5}$$

where W_{pre} and W_{post} are constants about the presynaptic and postsynaptic activity, A_+ and A_- characterize the extent to which synaptic changes depend on the current synapse weights. W_+ and W_- are respectively the time windows of the long-term potentiation (LTP) and the long-term depression (LTD) processes, which satisfy $\int_0^\infty W(l)\mathrm{d}l = 1$. Tuple $\langle W_{\text{pre}}, W_{\text{post}}, A_+(w_{ij}), A_-(w_{ij})\rangle$ defines a specific STDP rule.

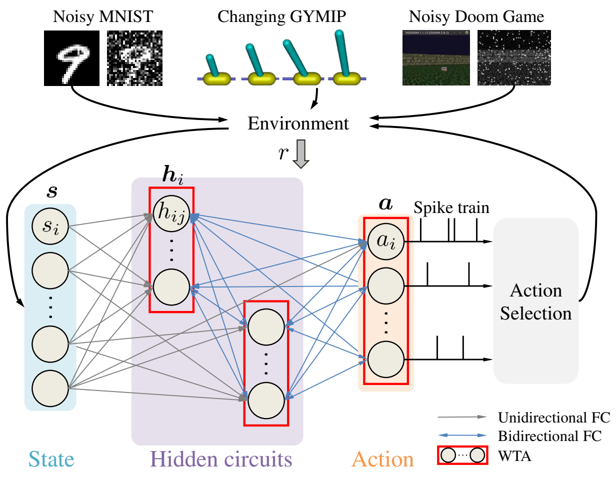


Fig. 2. SVPG is implemented by an RWTA Network. The environment represents different tasks of interest.

2.3 Markov Decision Process

We use the Markov decision process (MDP) for modeling RL tasks and adopt the notations from [31], i.e., $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$. At each time step t, the agent observes the state $s_t \in \mathcal{S}$, makes action $a_t \in \mathcal{A}$ according to its policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ and receives a scalar reward r_t. The environment transfers to a new state s_{t+1} according to the transition function $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$. The learning objective is to find a policy π that maximizes the expected return:

$$J(\pi) = \mathbb{E}_{\tau \sim P_\pi} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right], \tag{6}$$

where P_π denotes the trajectory distribution over policy π, τ is the sampled trajectory $\langle s_0, a_0, r_1, s_1, a_1, \ldots, r_T, s_T \rangle$ and T is the total length of the episode. We assume that the action space $\mathcal{A}$ consists of a finite number of actions.

Note that the spike time step l is different from the RL step t. We design that the SNN is simulated for a fixed number of spike time steps (e.g., 100) for each RL time step. The simulation generates the action a_t and uses the reward r_t to update the network parameters. The simulation is then reset for the next RL time step.

3 Method

This section first introduces the design of the RWTA network, then derives the policy inference and optimization based on the REINFORCE algorithm, and finally introduces the extension to the PPO algorithm.

3.1 The RWTA Network

SVPG is based on the recurrent winner-take-all (recurrent WTA, RWTA) network which is sketched in Figure 2. The RWTA network consists of some state neurons, some hidden WTA circuits, and one action WTA circuit. The firing probabilities of each state neuron encode one element of the state observation, while the firing states or probabilities of the action neurons can be used to generate the action decision. The network is fully connected, with all neurons from different circuits connected, but certain parts of the connections can be removed to create different network structures. The connections are symmetric, meaning the weight is shared by the two connected neurons. However, the connections that start from state neurons are unidirectional since the state neurons are not to be optimized.

The state neurons are denoted as s_i ($i = 1, \ldots, d_s$); the action neurons are a_i ($i = 1, \ldots, d_a$); the j-th neuron in the i-th hidden circuit is h_{ij} ($i = 1, \ldots, n_h$, $j = 1, \ldots, d_h$). Here d_s, d_h, and d_a are the sizes of the state observation and the WTA circuits, and n_h is the number of hidden WTA circuits. At each spike time step, each neuron has two properties: firing probability $q \in [0, 1]$ and binary firing status $v \in \{0, 1\}$. We use vectors to represent the values of groups of neurons, use h_i and h to denote the i-th hidden circuit and the entire set of hidden neurons, and use bold symbols with no subscript to denote all the neurons. For example, $\boldsymbol{q}_{h_i} := [q_{h_{i1}}, \ldots, q_{h_{id_h}}]^{\mathrm{T}}$, $\boldsymbol{v}_h := [\boldsymbol{v}_{h_1}^{\mathrm{T}}, \ldots, \boldsymbol{v}_{h_{n_h}}^{\mathrm{T}}]^{\mathrm{T}}$, and $\boldsymbol{q} = [\boldsymbol{q}_h^{\mathrm{T}}, \boldsymbol{q}_a^{\mathrm{T}}, \boldsymbol{q}_s^{\mathrm{T}}]^{\mathrm{T}}$. The total number of neurons is $N = n_h d_h + d_a + d_s$. The learnable parameters in the network are denoted as $\boldsymbol{W} \in \mathbb{R}^{N \times N}$ for the synapse weights and $\boldsymbol{b} \in \mathbb{R}^N$ for the self-activation parameters; the columns and rows of $\boldsymbol{W}$ are arranged by h, a, s and $\boldsymbol{b}$ is arranged according to h, a, s. Note that the state neurons have zero intrinsic excitabilities, i.e., $\boldsymbol{b}_s = \boldsymbol{0}$. We use θ to refer to the parameters of the policy, i.e., $\theta = \langle \boldsymbol{W}, \boldsymbol{b} \rangle$.

3.2 Policy Inference

3.2.1 *The definition of policy function*

We set the RL policy to be a probability distribution over the action space which is assumed to consist of a finite number of actions, and define it with an energy function $E(\boldsymbol{v})$:

$$\pi(\boldsymbol{v}_a|s) = \sum_{\boldsymbol{v}_h} p(\boldsymbol{v}_a, \boldsymbol{v}_h|s), \tag{7}$$

$$p(\boldsymbol{v}_a, \boldsymbol{v}_h|s) := \frac{1}{Z(s)} \exp\{E(\boldsymbol{v})\}, \tag{8}$$

$$E(\boldsymbol{v}) := \boldsymbol{v}^{\mathrm{T}} \boldsymbol{W} \boldsymbol{v} + \boldsymbol{b}^{\mathrm{T}} \boldsymbol{v}, \tag{9}$$

where $Z(s) = \sum_{\boldsymbol{v}'_h, \boldsymbol{v}'_a} \exp\{E(\boldsymbol{v}')\}$ is the normalization. As shown in the above equations, the policy distribution is calculated as the marginal distribution of $\boldsymbol{v}_a$. Although the energy function is linear, the normalization operation makes the energy-based policy function capable of representing complex distributions [52]. This formulation makes the policy function similar to the SRM model, i.e., Eq.s (1) and (2). More importantly, as will be shown later, this formulation of policy is equivalent to the fixed point of the RWTA network.

The policy representation Eq. (7) is computable in principle. However, when the number of hidden neurons is large, it can be intractable in practice. To address this problem, we use mean-field inference to derive an approximation $\hat{p}(\boldsymbol{v}_a, \boldsymbol{v}_h|s)$ of the probability function of action-hidden states $p(\boldsymbol{v}_a, \boldsymbol{v}_h|s)$.

3.2.2 *Validity of policy approximation*

The above approximation can induce a change to the expected return. Before diving into the details of the approximation, we first analyze the relationship between the approximation and the change. To do this, we equivalently transform the original objective $J(\pi)$ into $\log J(\pi)$. We use

τ_+ to denote the trajectories in which hidden states $\boldsymbol{v}_h$ are incorporated with actions. We use est to refer to the probabilities or distributions of trajectories under the approximated policy function. Then we can get the following lower bound of the objective when the policy approximation is applied:

$$\log J(\pi) = \log[\sum_{\tau_+} p_\pi(\tau_+) \sum_{t=0}^{T-1} \gamma^t r_t] \\ \geq \mathbb{E}_{\text{est}}[\log \sum_{t=0}^{T-1} \gamma^t r_t] - D_{\text{KL}}[p_{\text{est}}(\tau_+) \parallel p_\pi(\tau_+)]. \tag{10}$$

For deterministic environments, target $\mathbb{E}_{\text{est}}[\log \sum_{t=0}^{T-1} \gamma^t r_t]$ is equivalent to $\mathbb{E}_{\text{est}}[\sum_{t=0}^{T-1} \gamma^t r_t]$, which is the original expected return; in stochastic environments, the equivalence depends on the transition function P. When the equivalence holds, the above Eq. (10) indicates that, by minimizing the KL divergence between the approximated function $\hat{p}(\boldsymbol{v}_a, \boldsymbol{v}_h|s)$ and the original function $p(\boldsymbol{v}_a, \boldsymbol{v}_h|s)$, we can maximize the lower bound of the original expected return.

3.2.3 Policy mean-field inference

We use a variational distribution $\hat{p}(\boldsymbol{v}_a, \boldsymbol{v}_h|s)$ to approximate $p(\boldsymbol{v}_a, \boldsymbol{v}_h|s)$, and assume that the firing states of all circuits are independent to each other. This leads to a decomposition of $\hat{p}$, i.e., $\hat{p}(\boldsymbol{v}_a, \boldsymbol{v}_h|s) := \hat{p}(\boldsymbol{v}_a|s)\hat{p}(\boldsymbol{v}_{h_1}|s)\cdots\hat{p}(\boldsymbol{v}_{h_{n_h}}|s)$, where $\hat{p}(\boldsymbol{v}_{h_1}|s) := \boldsymbol{q}_{h_1}^{\mathrm{T}}\boldsymbol{v}_{h_1}, \ldots, \hat{p}(\boldsymbol{v}_a|s) = \boldsymbol{q}_a^{\mathrm{T}}\boldsymbol{v}_a$.

By minimizing the KL divergence between $\hat{p}$ and p, i.e., $D_{\text{KL}}(s) \doteq D_{\text{KL}}[\hat{p}(\boldsymbol{v}_a, \boldsymbol{v}_h|s)\|p(\boldsymbol{v}_a, \boldsymbol{v}_h|s)]$, we get the following mean-field inference equation [53] for each hidden or action neuron i:

$$q_i = \frac{1}{Z(q_{G(i)})} \exp\{\boldsymbol{w}_{\text{row},i}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\text{col},i}^{\mathrm{T}}\boldsymbol{q} + b_i\}, \tag{11}$$

where $i = 1, \ldots, (n_h d_h + d_a)$, $G(i)$ is the set of indices of the neurons in the same circuit as neuron i, $Z(q_{G(i)}) = \sum_{j\in G(i)} \exp\{\boldsymbol{w}_{\text{row},j}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\text{col},j}^{\mathrm{T}}\boldsymbol{q} + b_j\}$, and $\boldsymbol{w}_{\text{row},i}$ and $\boldsymbol{w}_{\text{col},i}$ are respectively the i-th row and column of matrix $\boldsymbol{W}$ (in the shape of a column vector), which corresponds to the synapses connected to neuron i. b_i is the i-th element in vector $\boldsymbol{b}$.

To get the policy distribution $\pi(\boldsymbol{v}_a|s)$, which is approximated by $\boldsymbol{q}_a$, we can solve Eq. (11) to get $\boldsymbol{q}$ and then extract its elements corresponding to $\boldsymbol{q}_a$. Eq. (11) can be seen as an iteration process by regarding the $\boldsymbol{q}$ on the right side as a constant vector. In practice, one numerical method to get the solution $\boldsymbol{q}$ is to initialize $\boldsymbol{q}$ with random numbers and then repeat updating it with Eq. (11) until numeric convergence. Although there is no theoretical guarantee of convergence, we demonstrate in our experiments that it converges in most cases (see section 4.2).

3.2.4 Policy inference with RWTA network

Now we show that the fixed point of the RWTA network equals the approximated policy inference above. That is, the iterative method above for policy inference can be implemented with the RWTA network.

We assume that the internal inhibitory neuron in the WTA circuits makes the overall firing rates of the network (excluding the state neurons) a constant value $\hat{\rho} \in (0, 1)$. With this assumption, we let the firing probabilities encode $\rho_i(l) = \hat{\rho} q_i$. Then the policy inference function Eq. (11) is transformed to

$$\rho_i = \hat{\rho}\exp\{\boldsymbol{w}_{\text{row},i}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\text{col},i}^{\mathrm{T}}\boldsymbol{q} + b_i \\ - \log \sum_{j\in G(i)} \exp\{\boldsymbol{w}_{\text{row},j}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\text{col},j}^{\mathrm{T}}\boldsymbol{q} + b_j\}\}. \tag{12}$$

Then, consider the neuron model Eqs. (1)and (2) in the RWTA network, we assign w_j with the synaptic weights $w_{ij} + w_{ji}$, and design $\kappa(y)$ such that $\int_0^\infty \kappa(y)\mathrm{d}y = 1/\hat{\rho}$. This transforms the probability values $\boldsymbol{q}$ in Eq. (12) into membrane potential $u(l)$, leading to the following spike-based inference function

$$\rho_i(l) = \hat{\rho}\exp\{u_i(l) - \log \sum_{j\in G(i)} \exp(u_j(l))\}, \\ u_i(l) = \sum_{j\in N(i)} w_{ij} \int_0^\infty \kappa(y) S_{ij}(l-y)\mathrm{d}y + b_i. \tag{13}$$

Eq. (13) shows the way the RWTA network iterates its membrane potentials and firing probabilities. When a fixed point is reached, the firing probabilities give the solution to the policy inference function Eq. (11). This shows that the RWTA network designed above can perform the approximated policy inference. Note that it is biologically plausible as it conforms to the definition of the SRM neuron model.

3.3 Policy Optimization

The policy optimization concerns the update of network parameters θ and relies on a base RL algorithm. Here we select REINFORCE as the base algorithm, because it is the base of many popular algorithms like A2C and PPO, and it has a simple policy gradient formulation which can simplify the derivation of our method. We derive the learning method and build its relationship to the R-STDP framework. Then we extend the method to other base RL algorithms like PPO.

3.3.1 Policy optimization for REINFORCE

In REINFORCE, the policy gradient is calculated according to the following equation [31]:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T-1} \gamma^t r_t \sum_{k=0}^{T-1} \nabla_\theta \log \pi_\theta(a_k|s_k)], \tag{14}$$

which contains $\nabla_\theta \log \pi_\theta(a_k|s_k)$, differential of logarithm of the policy function. Based on the policy approximation $\hat{p}(\boldsymbol{v}_a, \boldsymbol{v}_h|s)$ in previous subsection, this differential stands for $\nabla_\theta \log(\boldsymbol{q}_{ha})$. According to the policy inference function Eq. (11), this differential can be calculated as shown in the following theorem.

Theorem 1. *(Precise optimization rule) The* ***precise differential*** *of $\boldsymbol{q}_{ha}$ to a certain synapse weight w_{jk} and the self-activation parameter b_j is*

$$\frac{\partial \boldsymbol{q}_{ha}}{\partial w_{jk}} = \boldsymbol{M}(\boldsymbol{U}_{jk} + \boldsymbol{U}_{kj})\boldsymbol{q} + \boldsymbol{M}(\boldsymbol{W} + \boldsymbol{W}^{\mathrm{T}})\frac{\partial \boldsymbol{q}}{\partial w_{jk}}, \\ \frac{\partial \boldsymbol{q}_{ha}}{\partial b_j} = \boldsymbol{M}\boldsymbol{b} + \boldsymbol{M}(\boldsymbol{W} + \boldsymbol{W}^{\mathrm{T}})\frac{\partial \boldsymbol{q}}{\partial b_j}, \tag{15}$$

where $\boldsymbol{M} = \mathrm{diag}(\boldsymbol{q}_{ha})[-\boldsymbol{G}_{ha}\mathrm{diag}(\boldsymbol{q}) + \boldsymbol{D}_{\mathrm{sel}}]$, $\boldsymbol{G}_{ha}$ is a logical matrix with shape $(n_h d_h + d_a) \times N$ where 1 elements indicate the two neurons (column index and row index) are in the same WTA circuit, $\boldsymbol{D}_{\mathrm{sel}}$ is a logical matrix that selects the first $(n_h d_h + d_a)$ elements in a vector with length N, i.e., $\boldsymbol{D}_{\mathrm{sel}} = [\boldsymbol{I}_{(n_h d_h + d_a)} \quad \boldsymbol{O}_{(n_h d_h + d_a) \times d_s}]$, and $\boldsymbol{U}_{jk}$ is a logical matrix with shape $N \times N$ where only the jk-th element is 1.

Proof. The proof can be seen in Appendix A.1. □

Theorem 1 reveals that the required differential can be obtained by solving the matrix equations Eq. (15). However, this involves the calculation of the pseudo-inverse of $\boldsymbol{M}(\boldsymbol{W}+\boldsymbol{W}^{\mathrm{T}})$, the shape of which is $(n_h d_h + d_a) \times N$. Therefore the computational complexity is over $O((n_h d_h + d_a)^3)$ which can be intractable in practice when the number of hidden and action neurons is large.

Therefore we turn to obtain an approximated solution of Eq. (11). As will be shown later, this approximation can still get satisfying results in our experiments; it can also be implemented in the R-STDP framework so has the advantage of being biologically plausible. The idea for the approximation is to regard the $\boldsymbol{q}$ on the right side of the policy inference function Eq. (11) as a constant on the network parameters. By doing so, the differential only concerns the last step in the inference process, where the status of each neuron is only affected by its neighboring neurons. Thus the differential on a certain connection or neuron only depends on information from the connected neurons, which makes possible the link between the local rules and the global objective. The result is presented in the following theorem.

Theorem 2. *(Approximate optimization rule) The **approximate differentials** of firing rate q_i with respect to $\boldsymbol{W}$ and $\boldsymbol{b}$ are:*

$$\begin{aligned}\frac{\partial \log(q_i)}{\partial \boldsymbol{W}} =&(\boldsymbol{U}_{i:}\mathrm{diag}(\boldsymbol{q}) + \mathrm{diag}(\boldsymbol{q})\boldsymbol{U}_{:i}) \\ &- \mathrm{diag}(\boldsymbol{q})(\boldsymbol{U}_{G(i):} + \boldsymbol{U}_{:G(i)})\mathrm{diag}(\boldsymbol{q}), \\ \frac{\partial \log(q_i)}{\partial \boldsymbol{b}} =&\boldsymbol{u}_i - \mathrm{diag}(\boldsymbol{q})\boldsymbol{u}_{G(i)},\end{aligned} \tag{16}$$

where $i \in \{1, \ldots, (n_h d_h + d_a)\}$, $\boldsymbol{U}$ is a $N \times N$ logical matrix and $\boldsymbol{u}$ is a length-N logical vector, whose subscripts indicates the positions of elements with value 1. $G(i)$ is the set of indices of neurons in the same circuit as neuron i; ":" means the entire row/column.

Proof. The proof can be seen in Appendix A.2. □

According to Theorem 2, at the last step of each simulation for a certain RL time step t, given the firing state $\boldsymbol{v}$ of the RWTA network, we can obtain the corresponding REINFORCE policy gradient

$$\nabla J(\pi) = \sum_t \gamma^t r_t [\sum_{i=1}^{n_h} \boldsymbol{v}_{h_i}^{\mathrm{T}} \nabla(\log \boldsymbol{q}_{h_i}) + \boldsymbol{v}_a^{\mathrm{T}} \nabla(\log \boldsymbol{q}_a)], \tag{17}$$

where $\nabla \log \boldsymbol{q}_{h_i}$ and $\nabla \log \boldsymbol{q}_a$ are respectively the vectors of $\nabla \log q_{h_{ij}}$ and $\nabla \log q_{a_i}$.

3.3.2 Policy optimization with R-STDP

Now we show how this policy gradient can be implemented with R-STDP. Specifically, this means to design a set of $\langle W_{\mathrm{pre}}, W_{\mathrm{post}}, A_+(w_{ij}), A_-(w_{ij})\rangle$ in the R-STDP framework. We make the following settings to the R-STDP for two arbitrarily connected neurons i and j.

$$\langle W_{\mathrm{pre}}, W_{\mathrm{post}}, A_+(w_{ij}), A_-(w_{ij})\rangle = \langle v_i, v_j, -1/\hat{\rho}, -1/\hat{\rho}\rangle, \tag{18}$$

In practice, when the number of spike time steps in the simulation is large enough, the frequency of spikes in a spike train S_i reflects the firing probability ρ_i, i.e., $\mathbb{E}[S_i(l)] = \rho_i$ and $\mathbb{E}[\int_0^\infty A_+(w_{ij})S_i(l-y)\mathrm{d}y] = A_+(w_{ij})\rho_i$. Then, we have the following transformed formulation of our R-STDP rule:

$$\mathbb{E}[R(l)\mathrm{STDP}(l)] = R'[\rho_j(v_i - \rho_i/\hat{\rho}) + \rho_i(v_j - \rho_j/\hat{\rho})], \tag{19}$$

where R' is a signal about the environment reward for the considered simulation period. Note that, for the self-excitation parameter b, it can be regarded as the weight of a connection from an always-firing neuron and that the post-synaptic part of the STDP rule is omitted. The corresponding learning rule is $\Delta b_i = R'[v_i - \rho_i/\hat{\rho}]$.

Then, for the policy gradient Eq. (16, 17) which is derived from the global objective, we can reorganize them according to the network parameters as follows:

$$\begin{aligned}\frac{\partial J(\pi)}{\partial w_{ij}} =&\sum_t \gamma^t r_t [q_i(v_j - q_j) + q_j(v_i - q_i)], \\ \frac{\partial J(\pi)}{\partial b_i} =&\sum_t \gamma^t r_t (v_i - q_i).\end{aligned} \tag{20}$$

As shown, the two equations Eq. (19) and Eq. (20) are equivalent. By using Monte-Carlo sampling methods, we can make R' equal to $\sum_t \gamma^t r_t$. By scaling the optimization step size with $\hat{\rho}$, we can remove the difference in the overall firing rate $\hat{\rho}$. This means that the R-STDP rule defined in Eq. (18) can represent the approximated policy gradient on the RWTA network.

So far we derive a variational policy gradient method where inference and optimization are implemented with the spiking RWTA network and an R-STDP rule. We name it spiking variational policy gradient (SVPG).

3.4 Practical Considerations

There may be two problems with SVPG in practical application. (1) The simulation of spike trains in the RWTA network can be computationally expensive, particularly for general devices like GPU. (2) SVPG is derived for the REINFORCE algorithm, which is not efficient and is not popularly used in recent RL studies. Here we provide solutions to these two potential problems.

3.4.1 Rate-based approximation

For the computational cost problem, we propose a rate-based approximation of SVPG. In the approximation, the evolution of neurons' firing probabilities is directly calculated by the policy inference function Eq. (11), i.e., without the intermediate simulation of the spike trains. In this way, the computational cost can be reduced. However, this approximation also removes the random noises in firing probabilities caused by spike trains, which could be important to the overall performance. For this deficiency, we add Gaussian noises (with standard deviation $\sigma = 0.02$) to the firing probability values in each iteration of the firing probabilities. As will be shown later (section 4.2.2), this rate-based approximation can significantly reduce the computational cost while producing similar training and perturbation results to the original implementation.

3.4.2 Extension to other base RL algorithms

The RL field has seen many advances in algorithms that bring improvements to training efficiency, scalability, etc. Extending SVPG to these RL algorithms can facilitate the test or application to more challenging scenarios. Here we consider the PPO-clip algorithm [38], which has been widely used in RL studies and is generally considered faster and better at solving complex tasks like DOOM than REINFORCE. We also consider the extension to value-based RL algorithms, which are another major branch besides policy gradient.

For value-based algorithms like DQN, the network is required to output a number of state-action values [54]. The firing rates of the action neurons can be used to approximate their firing probabilities q_{a_i}, and then transformed to the state-action value with a mapping like $Q_{a_i} = \tan\{q_{a_i}\pi - \pi/2\}$. Suppose a loss function $\mathrm{Loss}(Q_{a,i})$ on the state-action value is defined in the base RL algorithm, we can decompose its differential into two parts according to the chain rule $\partial \mathrm{Loss}(Q_{a_i})/\partial\theta = \frac{\partial \log(q_{a_i})}{\partial\theta} \cdot \frac{\partial \mathrm{Loss}(Q_{a_i})}{\partial \log(q_{a_i})}$; the first part has been derived in Eq. (16); the second part can be calculated in practice using deep learning libraries such as PyTorch [55].

For the PPO-clip algorithm, the learning target is [38]

$$J(\pi_\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)A_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right], \quad r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t), \tag{21}$$

where π_θ is the current policy and $\pi_{\theta_{\text{old}}}$ is an old policy with checkpoint parameter θ_{old}; ϵ is a hyperparameter. Similar to the value function representation, the differential of the policy distribution can be transformed into the one we derived earlier: $\frac{\partial q_i}{\partial\theta} = \frac{\partial \log(q_i)}{\partial\theta} \cdot q_i$. Note that there is a difference between conventional ANNs and the RWTA network when dealing with θ_{old}. The differential for the RWTA network, i.e., Eq. (20) requires the firing states of the neurons which can be different in different simulations. Thus the checkpoint θ_{old} needs to include both the network parameters and the firing states $\boldsymbol{v}$; when updating the network with Eq. (20), the v values are from the checkpoint, and the q values are from the current policy instantiation. This extension reduces the biological plausibility of the SVPG method since it uses information from a previous state of the network, however, this is inevitable for most base RL algorithms that use the target network technique [54]. A sketch of the SVPG algorithm for PPO-clip is given in Algorithm 1.

Algorithm 1 SVPG for PPO-clip

Parameter: Discount factor γ. Training episode number N_{epi}. Inference iteration number N_{iter}. Learning rate η. PPO epoch number N_{PPO}. Network shape n_h, d_h, d_a, d_s.
Output: RWTA Network parameter θ.

```
1:  Initialize $\theta$ to zero. Initialize the critic network.
2:  for Episode = 1, ..., $N_{\text{epi}}$ do
3:      Clear memory buffer $\mathcal{D}$.
4:      for Training step $t$ = 1, ..., $T$ do
5:          Observe and encode state $s_t$.
6:          Randomly initialize $\boldsymbol{q}_a$ and $\boldsymbol{q}_h$ and normalize them
            at circuit-level.
7:          Iterate Eq. (13) for $N_{\text{iter}}$ spike time steps. {Inference}
8:          Use $\boldsymbol{v}_a$ to generate $a_t$. Perform $a_t$, observe reward
            $r_t$ and new state $s_{t+1}$.
9:          Store $\langle s_t, a_t, r_t, s_{t+1}, \boldsymbol{q}, \boldsymbol{v}\rangle$ into $\mathcal{D}$.
10:     end for
11:     Get data from $\mathcal{D}$. Backtrack reward $R = \sum_t \gamma^t r_t$.
12:     Update critic network. Use critic to generate advan-
        tage value $A$.
13:     Store checkpoint $\theta_{\text{old}}$, $\boldsymbol{v}_{\text{old}}$ .
14:     for PPO epoch num = 1, ..., $N_{\text{PPO}}$ do
15:         Update $\theta$ using $A$, $\mathcal{D}$, $\theta_{\text{old}}$, $\boldsymbol{v}_{\text{old}}$ and Eq. (20), Eq. (21).
            {Optimization}
16:     end for
17: end for
```

4 Experiments

We apply SVPG to different kinds of RL tasks and compare it to representative methods of other types. This section will first introduce the task settings and the compared methods. Next is the empirical verification of assumptions made in the theory part. Then comes the task performances and results of perturbation tests. The rear of this section provides ablation tests and visualizations. The code for all the tasks and methods can be found at https://github.com/yzlc080733/SVPG2023.

4.1 Tasks and Compared Methods

We use five tasks in our experiments: reward-based MNIST classification, Gym InvertedPendulum, ViZDoom Health-Gathering [56], AI2THOR navigation [35], [36], and robot-arm reaching (built on PyRep [57] and CoppeliaSim [37]). In the following texts and figures, we use MNIST, GYMIP, DOOM, AI2THOR, and ROBOTARM to refer to these tasks.

- In MNIST, the objective is to select the correct label given the image input. The state is a vector of length 784 which is reshaped from the image; the action space corresponds to the 10 labels; the reward is $\{-1, +1\}$ corresponding to wrong or correct predictions.
- In GYMIP, the objective is to balance a pendulum for as long as possible. The maximum episode length is set to 200; the state is a length-4 vector of physical variables which are mapped to the range of $[0, 1]$; the action space is $\{-3, -1.5, 0, 1.5, 3\}$ meaning the force on the cart; the reward is always $+1$.
- In DOOM, the objective is to navigate and pickup boxes to survive as long as possible. The maximum episode length is set to 525; the state is a vector of length 4800 reshaped from the 80×60 first-person

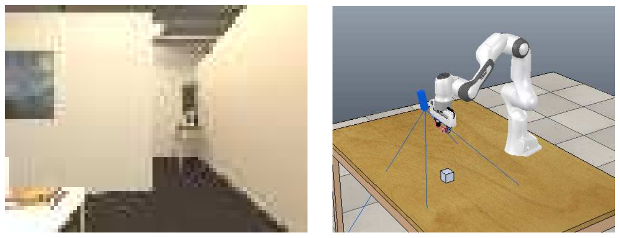


Fig. 3. (a) Example observation in AI2THOR. (b) Environment overview in ROBOTARM.

gray-scaled visual observation; the action space consists of 5 actions; the reward depends on the agent's health value; an example state observation is shown in Figure 8a.

- In AI2THOR, the objective is to navigate to a television in a realistic room. The maximum episode length is set to 200; the state is based on first-person vision and is pre-processed in the same way as in DOOM. The action space consists of 5 actions; the reward depends on the target distance and whether an action is successful. The starting point of the robot is randomly sampled from 38 positions, among which 30 are for training and 8 are for testing. An example state observation is in Figure 3a.
- In ROBOTARM, the objective is to move the gripper of a robot arm to the cube on the desk. The maximum episode length is 60; the state is a vector of length 4096 reshaped from the 64×64 visual observation obtained at the position of the gripper; the action space consists of 4 horizontal movements and one "move down" movement. The reward is determined by the target distance. The position of the target object is randomly sampled from 50 positions, among which 45 are for training and 5 are for testing. An example environment overview is in Figure 3b.

The MNIST task is selected because it is a single-step RL task, making its performance less affected by the training efficiency and exploration strategy. The GYMIP task is selected because it is a standard task widely used in the literature [51], [58], [59]; also its state variables are unbounded, providing a good example of state mapping for SVPG. The DOOM task is selected because it involves a high-dimensional vision input and a long episode horizon which challenges the methods' overall capability. The AI2THOR and ROBOTARM are used to reflect the methods' applicability to real-world tasks. In particular, AI2THOR provides photo-realistic scenes [35] (lighting, texture, etc.) and simulates collisions and noises in the robot's movements. In addition, the robot needs to generalize to new starting points (AI2THOR) or target positions (ROBOTARM).

Note that an encoding of the state observation is necessary to get the firing probabilities of the state neurons, because the latter is constrained to range $[0, 1]$. In MNIST, DOOM, AI2THOR, and ROBOTARM, the elements in state observations have limited values so can be linearly mapped to $[0, 1]$. In GYMIP, the state values are unbounded; therefore we first train an ANN to get samples to estimate the range and then clip and map to $[0, 1]$. More details about the task settings, including state encoding, reward function, and action space are in Appendix C.1.

We select three representative learning methods for comparison. The first one serves as a conventional approach in deep RL; the other two are usual approaches in SNN-based RL.

- *BP* [60] (backpropagation) on a three-layer multi-layer perceptron with the ReLU function for the hidden layers which is a conventional baseline ANN model.
- *ANN2SNN* with the methods from [61] and code implementation from SpikingJelly [62]. This method is based on the training results from the BP method.
- Fast sigmoid *BPTT* (backpropagation through time) from [63]. The code implementation is based on snnTorch [64].

For these compared methods, we set the number of hidden layers to 1. In all the comparisons, we use the same optimizer (RMSprop or Adam), discount factor γ, number of hidden neurons, and base RL algorithm. We tune the learning rate and entropy ratio for each method and report the result with the best zero-perturbation testing performance. Other implementation details are introduced in Appendix C.2.

We also considered two local-learning-rule-based methods. The first is a hybrid method of STDP and R-STDP [65], implemented on SpykeTorch [66], and designed for MNIST. We refer to it as *Mozafari et al*. This method enables the training of deep networks by applying STDP to hidden layers. For a fair comparison, we changed its network structure to a multi-layer perceptron with the same shape as other compared methods and removed the difference of Gaussians filter. Other designs including latency encoding and adaptive learning rate are kept. The second is a local gradient-based optimization method [67], implemented for CartPole, an environment similar to GYMIP. We refer to it as *Aenugu et al*. This method uses generalized linear model as neurons and updates connection weights based only on the local spiking activity and the global reward information. We reduced the size of the hidden layer to make the comparison fair. The input encoding, sparse connection, critic model and voting mechanism are kept.

We notice that the RWTA network in our method is fully-connected so has more learnable parameters than other methods under the same number of hidden neurons. Therefore we add a variant *SVPG-shrink* with fewer hidden WTA circuits, of which the number of learnable parameters is equal to or less than the networks in other methods. We provide the details in Appendix C.2.

4.2 Assumption Verification

In the theory part, we made two assumptions. (1) In section 3.2.3, we assume that iterating the firing probabilities with the policy inference function Eq. (11) can reach numeric convergence. (2) In section "practical consideration" 3.4.1, we propose an approximated implementation of SVPG. Here we use empirical results to verify these assumptions.

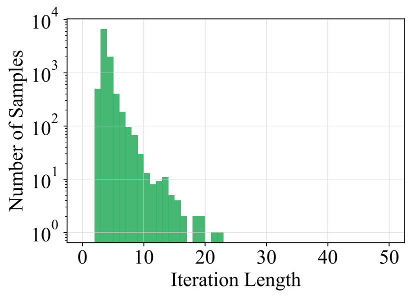


Fig. 4. Histogram of iteration lengths on MNIST.

4.2.1 Convergence verification

The policy inference part of SVPG, either rate-based or spike-based, relies on the iteration of the policy inference function Eq. (11). Here we use the MNIST task to empirically verify the convergence of the iteration process.

We monitor the firing probabilities of hidden and action neurons and set a stopping criterion for the iteration which is that the mean absolute changes in those probabilities in an iteration are smaller than 0.005, or that the iteration exceeds 50 steps. Note that this stopping criterion is also adopted in training rate-based SVPG. We feed the RWTA network with the testing images one by one and record their corresponding iteration lengths. The distribution of the iteration lengths is plotted in Figure 4. As shown, for all the tested images the iteration converges within 30 steps; also, most images correspond to an iteration length smaller than 10. These empirically verify that the policy inference function converges under most input cases.

4.2.2 Rate-based SVPG implementation

We compare the rate-based SVPG implementation with the original spike-based implementation on MNIST and GYMIP. The testing results with different strengths of perturbations to input and network parameters are shown in Figure 5. Details of perturbations are in Section 4.4. In Figure 5, the curves are averaged across 10 independent trainings, and the shades represent the standard deviation values. To save space, we put the complete set of figures in Appendix D.1.

As shown, the two implementations generate similar results under the tested perturbations, indicating that the rate-based implementation can be used as a replacement for spike train simulation. In the following experiments, we use the rate-based implementation to represent SVPG.

4.3 Task performances

We train different methods respectively on the five tasks. For the optimizer, we use Adam on MNIST, DOOM, AI2THOR, and ROBOTARM; we use RMSprop on GYMIP. This brings variances to the selection of the optimizer. For the base RL algorithm, in our previous paper [43], we used REINFORCE. Here we upgrade it to PPO-clip because of its popularity and training efficiency.

4.3.1 Zero-perturbation testing performances

The zero-perturbation testing performances are shown in Table 1. The values for MNIST, GYMIP, and DOOM are

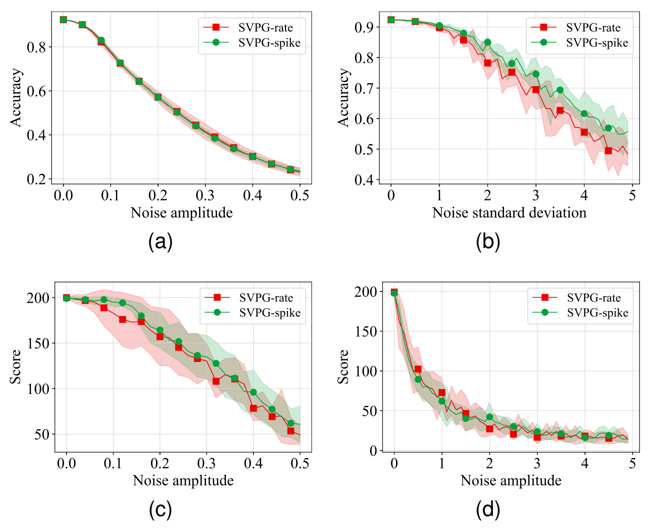


Fig. 5. Rate-based SVPG v.s. spike-based SVPG. (a) Input salt noise on MNIST. (b) Network Gaussian noise on MNIST. (c) Input uniform noise on GYMIP. (d) Network uniform noise on GYMIP.

from 10 independent trainings; the values for AI2THOR and ROBOTARM are from 3 independent trainings. The presented values are in the form of mean±standard deviation. For the first three tasks, a higher performance value is better; for the last two tasks, a lower performance value is better.

- On MNIST, the performance is measured by the testing accuracy. SVPG performs not as well as BP, BPTT, and ANN2SNN.
- On GYMIP, the performance is measured by the length of testing episodes and the optimum value is 200. SVPG achieves optimal performance.
- On DOOM, the performance is measured by the length of testing episodes and the optimal value is 525. SVPG achieves optimal performance.
- On AI2THOR, the performance is measured by the average number of steps the agent used to reach the target from different starting points. The optimal value is 24. SVPG achieves optimal performance.
- On ROBOTARM, the performance is measured by the averaged number of steps the agent used to reach the target from different starting points. SVPG achieves near-optimal performance.

These results indicate that SVPG with the RWTA network is able to solve image-input, long-horizon, and realistic RL tasks.

4.3.2 Computational costs

The computational costs are important for the practical applications of the methods. Here we provide results on the time complexity, space complexity, and sample efficiency. The *Mozafari et al.* R-STDP method and the *Aenugu et al.* method are not measured because their implementation [65], [67] do not support parallel processing of multiple samples, resulting in low space complexity and high time complexity.

(1) Time complexity. We measured the time required for the inference and optimization steps to reflect the time

TABLE 1
Zero-Noise Testing Performances on the 5 Tasks. △: Higher better. ▽: Lower better.

Tasks	SVPG	BP	BPTT	ANN2SNN	Mozafari et al.	Aenugu et al.
MNIST △	0.929±0.001	0.979±0.001	0.975±0.001	0.978±0.002	0.587±0.010	-
GYMIP △	200.00±0.00	200.00±0.00	198.18±3.68	200.00±0.00	-	195.11±7.30
DOOM △	525.00±0.00	525.00±0.00	394.98±151.14	525.00±0.00	-	-
AI2THOR ▽	24.00±0.00	24.00±0.00	82.77±82.89	24.00±0.00	-	-
ROBOTARM ▽	7.43±0.26	6.93±0.19	25.40±24.47	7.20±0.16	-	-

TABLE 2
Time Complexity on MNIST.

Time (ms)	SVPG-rate	SVPG-spike	BP	BPTT
Inference	3.26±0.28	677.21±13.50	**0.18±0.02**	8.51±0.17
Optimization	**1.40±0.17**	1.49±0.15	1.43±0.06	5.54±0.06

complexity. We selected the MNIST task for this test because it has a consistent batch size and that the high-dimensional state induces a large computational cost. Our implementation of the methods is all based on PyTorch and runs on the same machine, with NVIDIA T600 GPU. The results are shown in Table 2. The values are averaged across 500 inference/optimization steps.

As shown, the rate coding variant of SVPG is more than 100 times faster than the spiking variant, which supports that using the rate approximation can reduce computation costs. Since the inference stage of RWTA requires iterations, the cost is higher than BP in which the MLP only needs a forward propagation. Besides, the rate-based SVPG is faster than the BPTT method. For the optimization stage, SVPG is faster than all the compared methods; this is because SVPG is a local learning method. Future work could leverage the local learning property of SVPG and implement it with neuromorphic hardware to improve the inference speed..

(2) Space complexity. We measured the memory costs of each method to reflect their space complexity. Again, we selected the MNIST task because it offers a consistent episode length which enables fair comparison. Different from normal training, in this test, we set the program to only use one CPU thread and no GPU, so the memory usage includes all the variables a method creates. We used the `psutil` package for Python to get the memory usages of the program before initialization and after training for 20 episodes, and used their difference as the memory costs. The results are presented in Table 3. The values are the mean and standard deviation values of results from 10 independent runs.

As shown, the memory cost of the spiking variant of SVPG is slightly higher than the rate-based variant. This is mainly due to the extra spike train recording in policy inference. Nevertheless, the memory cost of the spike-based SVPG is only slightly higher than BP (17.4%) and much lower than BPTT (48.2%). This indicates that SVPG may not cause much difficulty for practical applications in terms of space complexity.

(3) RL sample efficiency. The total training time depends on both the time complexity (time per step) and the sample complexity (number of steps in RL training). Here

TABLE 3
Space Complexity on MNIST. (Total memory used in megabytes)

Method	SVPG-rate	SVPG-spike	BP	BPTT
Memory Cost	603.7±10.8	611.2±17.7	520.4±6.5	1180.5±27.5

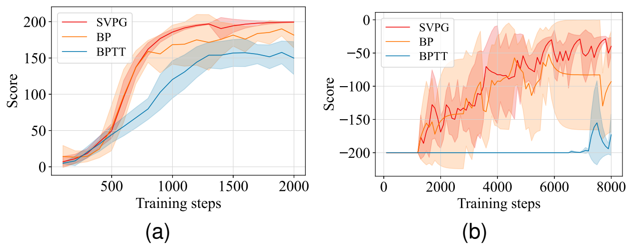


Fig. 6. Learning curves. (a) GYMIP. (b) AI2THOR.

we plot the learning curves to compare the RL sample efficiency of the methods. The 4 tasks with sequential decision-making are used (i.e., GYMIP, DOOM, AI2THOR, and ROBOTARM). Figure 6 presents the results on GYMIP and AI2THOR. The curves are the mean values of validation performances; the shades are the standard deviations. A full set of results and more details are in Appendix D.3.

Note that, for ease of viewing, we took the opposite value of the curves for AI2THOR and ROBOTARM. Specifically, the "score" equals (-1) times the step number. This makes the trends of these curves the same as those of the other three tasks. This is also done for the rest of the figures in this paper including the Appendix.

As shown, the learning curves of SVPG are close to BP and are higher than BPTT. This means that the sample efficiency of SVPG is similar to BP, which is a conventional ANN method widely used in recent RL studies. SVPG has the potential to be applied to scenarios where the efficiency of a traditional ANN is acceptable.

4.4 Perturbation Tests

It has been shown in many studies that SNNs (trained with ANN2SNN and BPTT methods) can exhibit better robustness to input and synapse weight noises [15], [41] and adversarial attacks [68] than ANNs. Here we perform tests on trained models to investigate whether SVPG can produce robustness.

We test three types of perturbations, namely input noise, network parameter noise, and environmental variation (in GYMIP). The input noises reflect inevitable sensor noises in the real world. The network parameter noise corresponds to parameter inaccuracies in neuromorphic hardware [69]. The

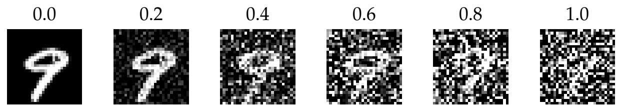


Fig. 7. Example input images with different strengths of Gaussian noises on the MNIST task. Standard deviation noted above images.

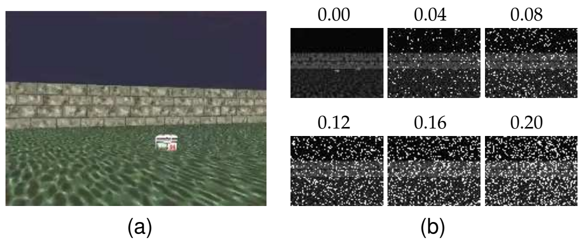


Fig. 8. Example state observation in the DOOM task. (a) Original observation. (b) Input salt noise; noise ratio noted above images.

environment variation in GYMIP represents differences in environment dynamics between the simulation and the real world.

- *Input noise* is independently added to each dimension of state observations. For MNIST, DOOM, AI2THOR, and ROBOTARM, Gaussian, salt, salt&pepper, and Gaussian&salt noises are considered. For GYMIP, Gaussian and uniform noises are considered. Some illustrations of the MNIST and DOOM task are shown in Figure 7, 8.
- *Network parameter noise* is independently added to each learnable parameter in the policy networks. Gaussian and uniform noises are adopted. Considering that different parts of the trained networks may have different scales of parameters, we divide the parameters into groups and normalize the noise using the mean absolute values within groups. For the RWTA network, the synapse weights $\boldsymbol{W}$ are divided according to the types of neurons connected, e.g., connections between state neurons and action neurons; the intrinsic excitability $\boldsymbol{b}$ forms one group. For the layered networks in other compared methods, the parameters are divided by layers and weight/bias. Note that this type of noise is regenerated for each testing episode.
- *Environmental variations in GYMIP*. In the GYMIP task, the optimal policy relies on the length and thickness of the pendulum. In training, we set $\langle$length=1.5, thickness=0.05$\rangle$. In testing, we change the length to be in the range of $[0.5, 4.9]$ and thickness of $[0.02, 0.30]$. Figure 9 illustrates these variations.

For each type of perturbation and each method, we select the hyperparameter (learning rate and entropy ratio) that generates the best zero-noise performance. When there are multiple hyperparameters that generate the same best zero-noise performance, we compare the performances under a certain level of perturbation. Note that on GYMIP there can be more than 10 hyperparameters that generate the best score; thus we use the average of all the hyperparameters with the best performance to better examine the robustness

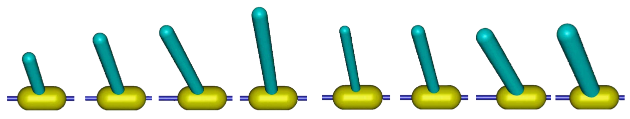


Fig. 9. Environmental variations in GYMIP testing. (a) Length. (b) Thickness.

of the method. Details are provided in Appendix C.2. In the following results Figure 10, 11 and 12, the curves come from the average of 10 independent trainings for MNIST, GYMIP, and DOOM, and 3 independent trainings for AI2THOR and ROBOTARM; the curves of GYMIP are also averaged across the selected hyperparameters; the shades represent the standard deviation values.

Note that the *Aenugu et al.* method contains an input encoder and a voting mechanism which are not included in other methods. These may affect the robustness to input and network parameter perturbations. Therefore, we only test its robustness to environment variations in the GYMIP task.

4.4.1 Input noises

The results are partially plotted in Figure 10. The full set of results is in Appendix D.2. In each test, we apply a range of different strengths of noises to the state inputs and test the agents' performances. For MNIST, the test uses all the samples from the testing images in the MNIST dataset. In GYMIP, DOOM, AI2THOR, and ROBOTARM, the test score for each training is the average value of 10 episodes. As the strength of the noises increases, the performance decreases; the speed of the decrease reflects the robustness of the methods. As shown, for the tested types of input noises on all three tasks, the performance of SVPG generally degrades slower than other methods. Instead of being robust on one task and sensitive on another (like BPTT which is the best on MNIST but the worst on GYMIP), SVPG displays a more consistent robustness across tasks. This shows that SVPG produces better inherent robustness to the tested input noises.

4.4.2 Network parameter noises

The results on MNIST, GYMIP, and DOOM are plotted in Figure 11. The experiment settings are the same as the test on input noises.

As shown, on all 5 tasks, SVPG achieves the slowest degradation of performance as the amplitude of noises increases. These indicate that SVPG generally produces better robustness to network perturbations than the compared methods.

Also note that in the above results on input and network parameter noises, *SVPG-shrink* and *SVPG* exhibit similar performances and robustness. This indicates that it is not the larger number of learnable parameters in the RWTA network that brings the difference in robustness.

4.4.3 Environmental variations in GYMIP

Results on variations in the pendulum's length and thickness in GYMIP are shown in Figure 12. When the shape of the pendulum deviates from the one in training, i.e., $\langle$length=1.5, thickness=0.05$\rangle$, the performance of all the

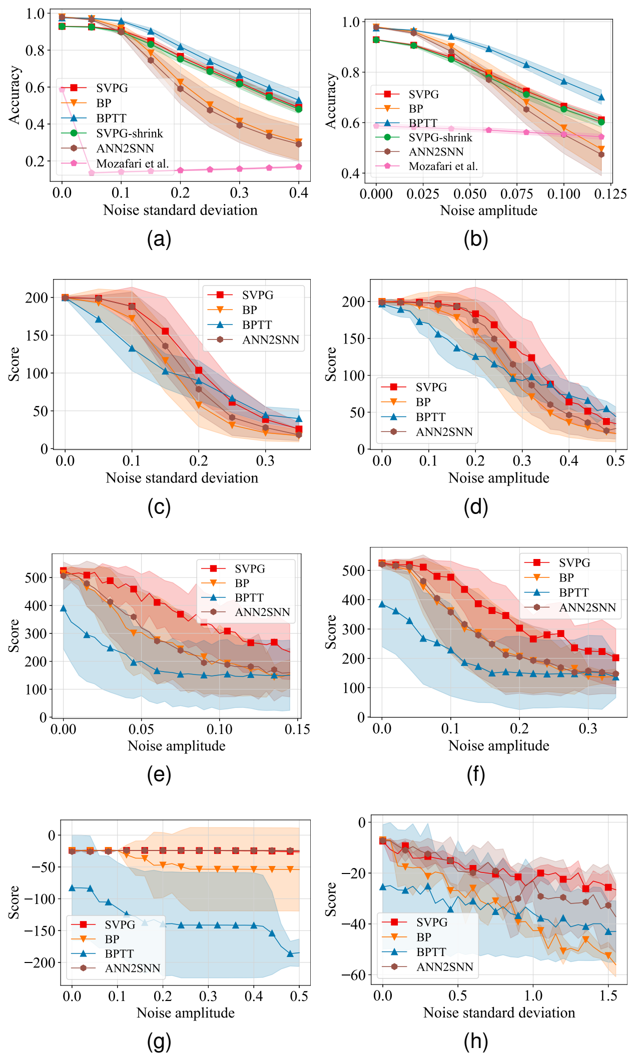


Fig. 10. Input noise tests. (a) MNIST Gaussian. (b) MNIST salt. (c) GYMIP Gaussian. (d) GYMIP uniform. (e) DOOM Gaussian & salt. (f) DOOM salt & pepper. (g) AI2THOR salt & pepper. (h) ROBOTARM Gaussian.

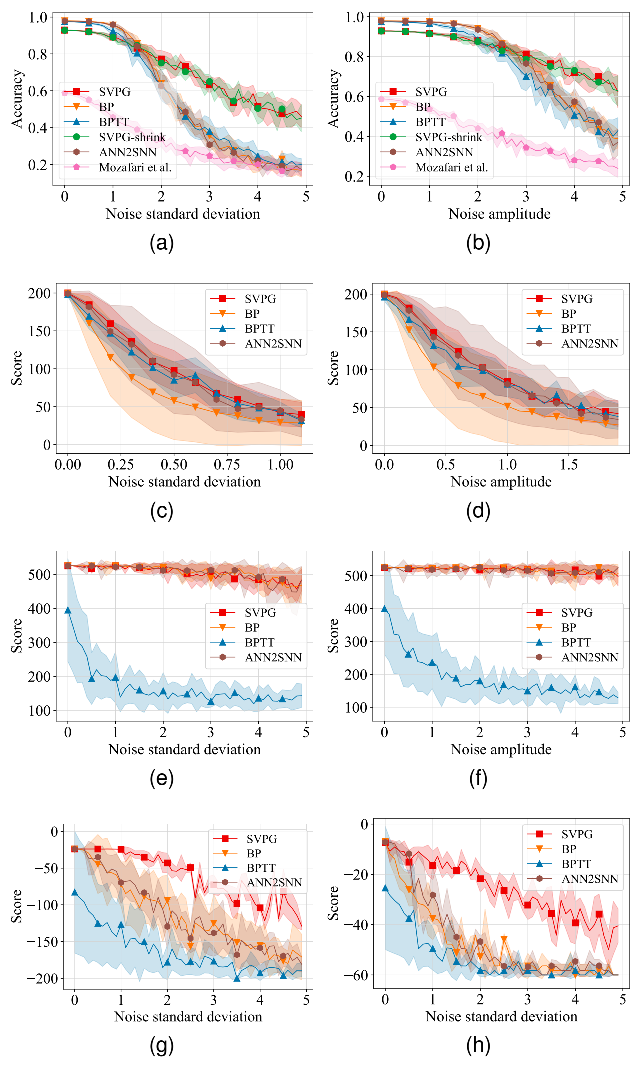


Fig. 11. Network parameter noise tests. (a) MNIST Gaussian. (b) MNIST uniform. (c) GYMIP Gaussian. (d) GYMIP uniform. (e) DOOM Gaussian. (f) DOOM uniform. (g) AI2THOR Gaussian. (h) ROBOTARM Gaussian.

compared methods degrades. For pendulum length, the performance of SVPG degrades slower than other methods. For pendulum thickness, SVPG is close to the best when the thickness is smaller than 0.15. These reveal that the policy trained using SVPG naturally adapts to a larger range of pendulums with different shapes.

So far we tested the robustness to input noises, network parameter noises, and environmental variations. As discussed, SVPG shows better robustness to most types of perturbations than the compared methods. We emphasize that in all these robustness tests the noises are only added in testing. Our results support the idea that SNNs could have better inherent robustness than conventional neural network models.

4.5 Ablation Study

We perform ablation tests to understand the effects of different parts of connections in the RWTA network. We use the MNIST task because the classification accuracy can better reflect the differences between performances.

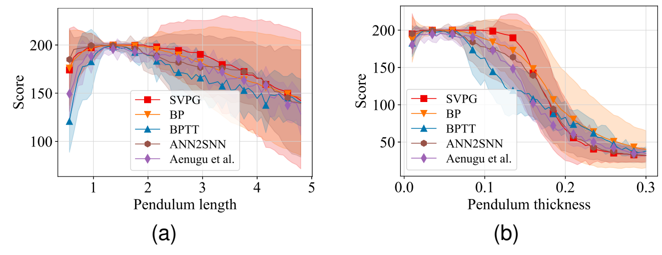


Fig. 12. Environmental variations in GYMIP. (a) Pendulum length. (b) Pendulum thickness.

4.5.1 *Effect of regional connection removal in training*

In this test, we remove different parts of connections in the RWTA network before training to investigate their contributions to learning. We divide the connections in the network according to the types of neurons they connect – connections between hidden and hidden neurons, hidden and action, state and action, and state and hidden. For conciseness, we use HH, HA, SA, and SH to represent these types. For example, we use "RM-HH" to refer to removing all connections

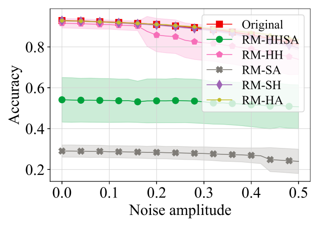


Fig. 13. Effects of connection removal in RWTA network in training on MNIST. (a) Input Pepper noise. (b) Network parameter uniform noise.

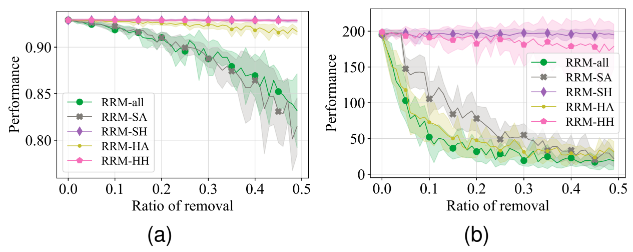


Fig. 14. Effects of connection removal in RWTA network in testing on (a) MNIST, (b) GYMIP. "RRM" means random removal of connections.

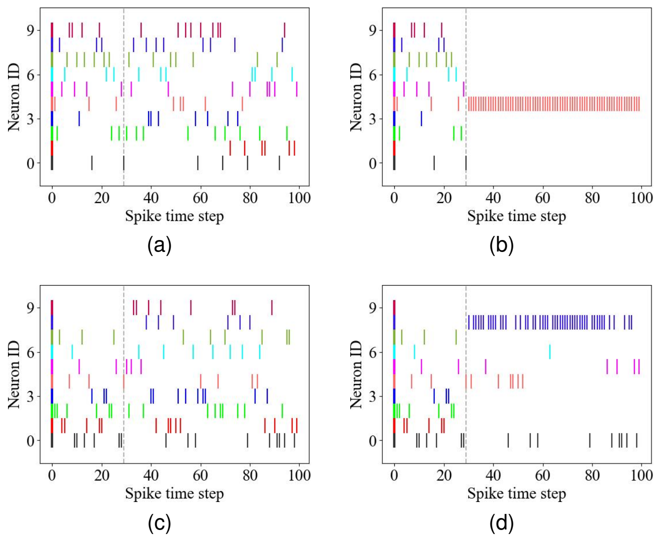


Fig. 15. Visualization of spike trains in WTA circuits on MNIST. (a, b) Action WTA circuit. (c, d) A hidden WTA circuit. (a, c) Randomly initialized RWTA network. (b, d) Trained RWTA network.

between hidden WTA circuits. By "original", we refer to the original RWTA network. We add perturbations to the input or network parameters during testing to better discriminate the performances. The results are shown in Figure 13.

As shown, for the tested perturbations, the original RWTA network produces the best performances. The settings of SH and HA produce performances similar to the original network. This indicates that SA connections play the most important role in training; after that is the HH connections.

We emphasize that the "RM-HHSA" setting corresponds to a three-layer structure in the RWTA network which is the same as [70]. As shown, the performance of this variation is obviously worse than the original network. This indicates that under the condition of the same number of hidden neurons, extending the model from layered to fully connected improves the training performance and robustness.

4.5.2 Effect of random connection removal in testing

In this test, we look at the effects of connection removal in testing to check their contribution to testing performance. We randomly remove a number of connections from a trained RWTA network, i.e., set the weight values to 0. We perform this test on connections in different parts of the network, and use HH, HA, SA, and SH to represent these types. We use ALL to refer to removing all four types of connections at the same time. As for measuring the strength of removal, we use the ratio of the number of removed connections to the number of original connections in the corresponding type.

This test is done on MNIST and GYMIP, and the results are plotted in Figure 14. As shown, as the ratio of removed connections increases, the testing accuracy drops in settings ALL, HA, and SA; it does not change significantly in settings SH and HH. This indicates that the testing performance depends more on HA and SA connections than on SH and HH connections. Among HA and SA connections, the effects of removing SA connections are stronger. This indicates that SA connections play a more important role than HA in testing performance. Although, in the previous part, HA connections do not contribute significantly to training, the results here show that HA connections do have effects on the testing performances.

4.6 Network Visualizations

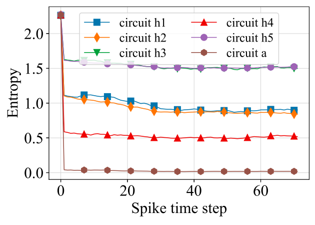
Here we provide visualizations of the RWTA network for a more intuitive understanding of its properties.

4.6.1 Firing process visualization on MNIST

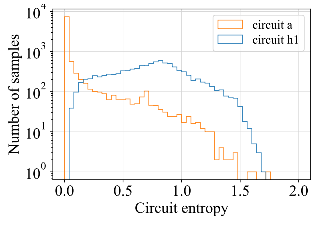
We collect the spike trains when a randomly selected image (No. 901) in the MNIST testing dataset is fed to the RWTA network, and plot them in Figure 15. The size of hidden and action WTA circuits is 10. In each plot, there are 10 spike trains each corresponding to one neuron in the WTA circuit. The length of simulation time and spike response window are respectively 100 and 30 spike time steps. In practice, we start updating the firing probabilities at the 29th spike time step, which is marked with dotted vertical lines.

As shown, in the RWTA network before training, the spike trains of the action circuit (Figure 15a) and hidden circuit (Figure 15c) are both noisy and evenly distributed across the neurons, even after some iterations of the firing probabilities. In contrast, in the trained RWTA network (Figure 15b, 15d), the spike trains quickly gather to a few neurons during the iteration of firing probabilities. This is consistent with the previous conclusion that the inference process converges quickly on MNIST.

We further calculate the entropy of the firing probability distribution in WTA circuits to measure their selectivity (after training). We first look at how the selectivity of the action WTA circuit and 5 randomly selected hidden WTA circuits changes during the inference process. In Figure 16a, circuits h1 to h5 are hidden circuits, and circuit a is the action circuit. As shown, in the policy inference process, the entropy of

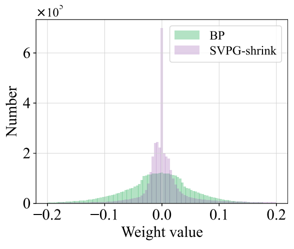


(a)

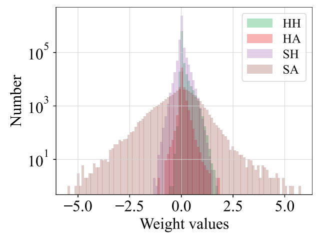


(b)

Fig. 16. Entropy values of WTA circuits on MNIST. (a) Change of entropy values of 5 hidden circuits and the action circuit during inference. (b) Distribution of entropy values of a hidden circuit and the action circuit on MNIST.



(a)



(b)

Fig. 17. Histograms of network weight values on MNIST. (a) Comparison of SVPG-shrink and BP. (b) Comparison of different types of connections in SVPG-shrink.

the circuits decreases sharply at the beginning and slowly afterward. The entropy of the action circuit is lower than the hidden circuits. We then look at the distribution of the entropy values when the MNIST testing set is fed to the network. As shown in Figure 16b, the distribution of the action WTA circuit concentrates at a value close to zero, while the distribution of the hidden circuit is smoother. These results indicate that, although the output of the RWTA network tends to converge to one neuron, the hidden part is less selective.

4.6.2 Firing process visualization on GYMIP

Different from MNIST, GYMIP involves changes in the environment states, which allows visualization of how the network dynamics changes with the environment. Due to the space limit, we put the visualizations on GYMIP in Appendix D.4 (Figure 39 and Figure 40). We found that the firing patterns of the neurons indeed change with the environment states. Besides, sometimes the output of the action WTA circuit changes gradually in the firing process. This indicates that the simulation of the firing process is meaningful.

4.6.3 Distribution of parameters in RWTA network on MNIST

Here we visualize the distribution of weight values in the networks and compare SVPG with BP. To make the comparison fair, we use the SVPG-shrink variant so the total number of weight values are close to that in BP. We plot the distribution of the weight values in a trained RWTA network and an MLP in Figure 17a. We also plot different types of weights in the RWTA network separately in Figure 17b. Note that the distribution is the stack of values across 10 independent trainings.

As shown, the ratio of zero weights in SVPG is greater than in BP. This indicates that SVPG tends to learn a more sparse network. In SVPG, SH and HH types of connections exhibit a sharp spike at value 0 in their weight distributions, indicating a more distinguished sparsity in those areas when compared to the other two types.

5 Related Works

There have been many studies on using SNNs as function approximators in RL. For example, the deep Q-learning algorithm has been implemented with an SNN based on the surrogate gradient method, which gains the "same performance level" as vanilla DQN in more than 10 Atari video games [71]. An SNN with course-scale approximated LIF neurons also uses a differentiable approximation of the gradients in solving the classic CartPole task [17]. The PopSAN method uses a rectangular function to approximate the gradient in the spatiotemporal backpropagation and solves several OpenAI Gym (mujoco) motor control tasks [72]. The SDDPG method also trains an SNN using the surrogate gradient method and achieves a better success rate than conventional DDPG in a laser-based navigation task [73]. In [15], a pre-trained DQN with ReLU activations is converted to an SNN which achieves similar performance as the original network in the Atari Breakout game. As shown, most of these successful applications use the surrogate gradient or ANN2SNN technique. The availability of the gradient brings flexibility to algorithm design and better performances. The main drawbacks are that the temporal information in state observations is not utilized by the SNNs and that the methods are not biologically plausible.

R-STDP methods feature biological plausibility because the learning rules use only local information. Starting from the original R-STDP [46], different variations of the modulation have been proposed, including TD-STDP [45], feedback-modulated TD-STDP [51], R-max [18], and hybrid [19]. Based on such methods, tasks including 2D goal-reaching [45], CartPole [45], [51], [70], LunarLander [51], and dynamic vision sensor-based lane keeping [23] have been solved. Nevertheless, these tasks are simpler than the ones that surrogate gradient-based and ANN2SNN-based methods can solve, like Atari games. One deficiency of these R-STDP methods is that the neuron models and STDP rules are not adapted to RL algorithms and network structures. In fact, most listed studies use a shallow network structure with less than three layers [23], [45], [51]. From this perspective, the EM-based learning rules in [29] and [70] are more promising since they could be applied to learn a deeper network with R-STDP; however, they are not derived for RL [29] or used a deep network [70].

The robustness of SNNs has been verified in many previous studies. A conversion from ANN to SNN has been shown to bring better robustness to input occlusions in Atari games [15] and noise in synaptic weights [41]. SNNs trained with BPTT can be more resilient to adversarial attacks [68] such as white-box attacks [74]. In reward-based MNIST classification and CartPole, a WTA-based SNN trained using R-STDP exhibits better robustness to input noise [70].

6 Conclusion and Discussion

Conclusion. We proposed SVPG, a policy gradient method based on R-STDP, to train an RWTA network. SVPG can solve multiple types of RL tasks, including a challenging DOOM task and two realistic robot control tasks. SVPG also produces policies that improve robustness to several types of perturbations. Ablation studies and visualizations show that different parts of the RWTA network indeed contribute to the final performance.

Discussion. This work focuses on the neural network model. Our results show that a brain-inspired model and brain-inspired learning method can solve difficult RL tasks, as well as generate better robustness to certain perturbations than compared methods. This supports the idea that brain-inspired models could by themselves produce better learning performances. Nevertheless, in practice, it is still promising to go beyond the constraint of biological plausibility and combine SVPG with other RL techniques, e.g., domain randomization and reward engineering. Besides, the time efficiency of the spiking-based SVPG is low on general computers. It would be promising to implement SVPG on neuromorphic hardware to improve the time efficiency. As for the space complexity, it would be beneficial to use sparse coding for the recording of spike trains and to set a smaller overall firing rate $\hat{\rho}$ for the RWTA network. At this stage, the rate-based SVPG should be better for applications as it achieves a computational efficiency similar to the conventional BP method.

Limitations. There are some limitations of SVPG.

- Current SVPG does not support continuous action spaces. This is because a limited number of action neurons is used to represent actions in the RWTA network. Adding further designs, e.g., Gaussian model of the action space, may be helpful for adaptation to continuous actions.
- SVPG is beyond the original definition of STDP [48], in which W_{pre} and W_{post} are constants with reference to neuronal activities, however in SVPG they are replaced by v_i and v_j. Nevertheless, SVPG only uses local signals so is still more biologically plausible than backpropagation-based methods.
- The PPO implementation is not fully biologically plausible. This is because PPO requires a policy checkpoint to constrain the size of learning steps; the checkpoint induces information beyond local firing signals. We would like to note that this problem is specific to the RL base algorithms and does not exist in the REINFORCE algorithm.

Future directions. Future directions may include theoretical analysis of the robustness results, more network structures (e.g., convolutional layers), more experiments in real-world control problems, and transplantation to neuromorphic hardware for real-life tasks.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv:1312.5602 [cs]*, Dec. 2013.

[2] A. Devo, G. Mezzetti, G. Costante, M. L. Fravolini, and P. Valigi, "Towards Generalization in Target-Driven Visual Navigation by Using Deep Reinforcement Learning," *IEEE Transactions on Robotics*, vol. 36, no. 5, pp. 1546–1561, Oct. 2020.

[3] M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, and Z. Wang, "Autonomous Navigation of Stratospheric Balloons Using Reinforcement Learning," *Nature*, vol. 588, no. 7836, pp. 77–82, Dec. 2020.

[4] T. Zhang, S. Guo, T. Tan, X. Hu, and F. Chen, "Adjacency Constraint for Efficient Hierarchical Reinforcement Learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 4, pp. 4152–4166, Apr. 2023.

[5] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2017, pp. 23–30.

[6] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning," in *International Conference on Learning Representations*, 2019.

[7] K. Wang, B. Kang, J. Shao, and J. Feng, "Improving Generalization in Reinforcement Learning with Mixture Regularization," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 7968–7978.

[8] C. Tessler, Y. Efroni, and S. Mannor, "Action Robust Reinforcement Learning and Applications in Continuous Control," in *Proceedings of the 36th International Conference on Machine Learning*. PMLR, May 2019, pp. 6215–6224.

[9] Z. Liu, J. Lu, J. Xuan, and G. Zhang, "Deep Reinforcement Learning in Nonstationary Environments With Unknown Change Points," *IEEE Transactions on Cybernetics*, pp. 1–14, 2024.

[10] T. Wunderlich, A. F. Kungl, E. Müller, A. Hartel, Y. Stradmann, S. A. Aamir, A. Grübl, A. Heimbrecht, K. Schreiber, D. Stöckel, C. Pehle, S. Billaudelle, G. Kiene, C. Mauch, J. Schemmel, K. Meier, and M. A. Petrovici, "Demonstrating Advantages of Neuromorphic Computation: A Pilot Study," *Frontiers in Neuroscience*, vol. 13, p. 260, Mar. 2019.

[11] H. Asgari, B. M.-N. Maybodi, M. Payvand, and M. R. Azghadi, "Low-Energy and Fast Spiking Neural Network For Context-Dependent Learning on FPGA," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 11, pp. 2697–2701, Nov. 2020.

[12] M. Chahine, R. Hasani, P. Kao, A. Ray, R. Shubert, M. Lechner, A. Amini, and D. Rus, "Robust flight navigation out of distribution with liquid neural networks," *Science Robotics*, vol. 8, no. 77, p. eadc8892, Apr. 2023.

[13] J. Wu, C. Xu, X. Han, D. Zhou, M. Zhang, H. Li, and K. C. Tan, "Progressive Tandem Learning for Pattern Recognition With Deep Spiking Neural Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 7824–7840, Nov. 2022.

[14] A. Suhaimi, A. W. H. Lim, X. W. Chia, C. Li, and H. Makino, "Representation Learning in the Artificial and Biological Neural Networks Underlying Sensorimotor Integration," *Science Advances*, vol. 8, no. 22, p. eabn0984, Jun. 2022.

[15] D. Patel, H. Hazan, D. J. Saunders, H. Siegelmann, and R. Kozma, "Improved Robustness of Reinforcement Learning Policies Upon Conversion to Spiking Neuronal Network Platforms Applied to Atari Games," *arXiv:1903.11012 [cs, stat]*, Aug. 2019.

[16] G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass, "A Solution to the Learning Dilemma for Recurrent Networks of Spiking Neurons," *Nature Communications*, vol. 11, no. 1, p. 3625, Dec. 2020.

[17] A. Yanguas-Gil, "Coarse Scale Representation of Spiking Neural Networks: Backpropagation Through Spikes and Application to Neuromorphic Hardware," in *International Conference on Neuromorphic Systems*. ACM, Jul. 2020, pp. 1–7.

[18] N. Frémaux and W. Gerstner, "Neuromodulated Spike-Timing-Dependent Plasticity, and Theory of Three-Factor Learning Rules," *Frontiers in Neural Circuits*, vol. 9, Jan. 2016.

[19] 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.

[20] G. Bellec, F. Scherr, E. Hajek, D. Salaj, R. Legenstein, and W. Maass, "Biologically Inspired Alternatives to Backpropagation Through Time for Learning in Recurrent Neural Nets." *CoRR*, vol. abs/1901.09049, 2019.

[21] A. Taherkhani, A. Belatreche, Y. Li, G. Cosma, L. P. Maguire, and

T. M. McGinnity, “A Review of Learning in Biologically Plausible Spiking Neural Networks,” *Neural Networks*, p. 20, 2020.

[22] H. Ghaemi, E. Mirzaei, M. Nouri, and S. R. Kheradpisheh, “BioLCNet: Reward-Modulated Locally Connected Spiking Neural Networks,” in *International Online & Onsite Conference on Machine Learning, Optimization, and Data Science*, 2022, pp. 564–578.

[23] 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 *IEEE International Conference on Robotics and Automation*, 2018, pp. 4725–4732.

[24] M. Pfeiffer and T. Pfeil, “Deep Learning with Spiking Neurons: Opportunities and Challenges,” *Frontiers in Neuroscience*, vol. 12, 2018.

[25] A. Sboev, D. Vlasov, R. Rybka, and A. Serenko, “Spiking Neural Network Reinforcement Learning Method Based on Temporal Coding and Stdp,” *Procedia Computer Science*, vol. 145, pp. 458–463, Jan. 2018.

[26] Z. Bing, Z. Jiang, L. Cheng, C. Cai, K. Huang, and A. C. Knoll, “End to End Learning of a Multi-Layered SNN Based on R-STDP for a Target Tracking Snake-Like Robot,” in *IEEE International Conference on Robotics and Automation*, 2019, pp. 9645–9651.

[27] D. J. Rezende, D. Wierstra, and W. Gerstner, “Variational Learning for Recurrent Spiking Networks,” in *Conference on Neural Information Processing Systems*, 2011, pp. 136–144.

[28] S. Guo, Z. Yu, F. Deng, X. Hu, and F. Chen, “Hierarchical Bayesian Inference and Learning in Spiking Neural Networks,” *IEEE Transactions on Cybernetics*, vol. 49, no. 1, pp. 133–145, Jan. 2019.

[29] H. Jang, N. Skatchkovsky, and O. Simeone, “VOWEL: A Local Online Learning Rule for Recurrent Networks of Probabilistic Spiking Winner-Take-All Circuits.” in *International Conference on Pattern Recognition*, 2020, pp. 4597–4604.

[30] Z. Yu, S. Guo, F. Deng, Q. Yan, K. Huang, J. K. Liu, and F. Chen, “Emergent Inference of Hidden Markov Models in Spiking Neural Networks Through Winner-Take-All,” *IEEE Transactions on Cybernetics*, vol. 50, no. 3, pp. 1347–1354, Mar. 2020.

[31] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018.

[32] Y. LeCun, C. Cortes, and C. J. Burges, “The MNIST Database of Handwritten Digits,” 1998.

[33] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI Gym,” *CoRR*, vol. abs/1606.01540, 2016.

[34] M. Wydmuch, M. Kempka, and W. Jaśkowski, “ViZDoom Competitions: Playing Doom from Pixels,” *arXiv:1809.03470 [cs, stat]*, Sep. 2018.

[35] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi, “AI2-THOR: An Interactive 3D Environment for Visual AI,” *arXiv:1712.05474 [cs]*, Dec. 2017.

[36] M. Deitke, W. Han, A. Herrasti, A. Kembhavi, E. Kolve, R. Mottaghi, J. Salvador, D. Schwenk, E. VanderBilt, M. Wallingford, L. Weihs, M. Yatskar, and A. Farhadi, “RoboTHOR: An Open Simulation-to-Real Embodied AI Platform,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2020, pp. 3161–3171.

[37] E. Rohmer, S. P. N. Singh, and M. Freese, “V-REP: A Versatile and Scalable Robot Simulation Framework,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1321–1326.

[38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” *CoRR*, vol. abs/1707.06347, 2017.

[39] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” in *International Conference on Learning Representations*, 2015.

[40] G. Hinton, N. Srivastava, and K. Swersky, “Overview of Mini-Batch Gradient Descent,” *Neural Networks for Machine Learning*, 2012.

[41] C. Li, R. Chen, C. Moutafis, and S. Furber, “Robustness to Noisy Synaptic Weights in Spiking Neural Networks,” in *International Joint Conference on Neural Networks*, Jul. 2020, pp. 1–8.

[42] C. A. Manrique Escobar, C. M. Pappalardo, and D. Guida, “A Parametric Study of a Deep Reinforcement Learning Control System Applied to the Swing-Up Problem of the Cart-Pole,” *Applied Sciences*, vol. 10, no. 24, p. 9013, Jan. 2020.

[43] Z. Yang, S. Guo, Y. Fang, and J. Liu, “Biologically Plausible Variational Policy Gradient with Spiking Recurrent Winner-Take-All Networks.” in *British Machine Vision Conference*, 2022, p. 358.

[44] B. Nessler, M. Pfeiffer, L. Buesing, and W. Maass, “Bayesian Computation Emerges in Generic Cortical Microcircuits through Spike-Timing-Dependent Plasticity,” *PLoS Computational Biology*, vol. 9, no. 4, 2013.

[45] 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, 2013.

[46] R. V. Florian, “Reinforcement Learning Through Modulation of Spike-Timing-Dependent Synaptic Plasticity,” *Neural Computation*, vol. 19, no. 6, pp. 1468–1502, Jun. 2007.

[47] Y. C. Yoon, “LIF and Simplified SRM Neurons Encode Signals Into Spikes via a Form of Asynchronous Pulse Sigma–Delta Modulation,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 5, pp. 1192–1205, May 2017.

[48] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014.

[49] J. N. J. Reynolds, B. I. Hyland, and J. R. Wickens, “A Cellular Mechanism of Reward-Related Learning,” *Nature*, vol. 413, no. 6851, pp. 67–70, Sep. 2001.

[50] D. E. Shulz, R. Sosnik, V. Ego, S. Haidarliu, and E. Ahissar, “A Neuronal Analogue of State-Dependent Learning,” *Nature*, vol. 403, no. 6769, pp. 549–553, Feb. 2000.

[51] S. Chung and R. Kozma, “Reinforcement Learning with Feedback-modulated TD-STDP,” *arXiv:2008.13044 [cs, stat]*, Aug. 2020.

[52] N. Heess, D. Silver, and Y. W. Teh, “Actor-Critic Reinforcement Learning with Energy-Based Policies,” in *European Workshop on Reinforcement Learning*, vol. 24, 2012, pp. 43–58.

[53] M. J. Wainwright and M. I. Jordan, “Graphical Models, Exponential Families, and Variational Inference,” *Foundations and Trends® in Machine Learning*, vol. 1, no. 1-2, pp. 1–305, 2008.

[54] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-Level Control Through Deep Reinforcement Learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[55] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Annual Conference on Neural Information Processing Systems*, vol. 32, 2019.

[56] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski, “ViZDoom: A Doom-Based AI Research Platform for Visual Reinforcement Learning,” in *IEEE Conference on Computational Intelligence and Games*, 2016, pp. 1–8.

[57] S. James, M. Freese, and A. J. Davison, “PyRep: Bringing V-REP to Deep Robot Learning,” *CoRR*, vol. abs/1906.11176, 2019.

[58] Â. G. Lovatto, T. P. Bueno, and L. N. Barros, “Analyzing the Effect of Stochastic Transitions in Policy Gradients in Deep Reinforcement Learning,” in *Brazilian Conference on Intelligent Systems*, Oct. 2019, pp. 413–418.

[59] R. Özalp, N. K. Varol, B. Taşci, and A. Uçar, “A Review of Deep Reinforcement Learning Algorithms and Comparative Results on Inverted Pendulum System,” in *Machine Learning Paradigms: Advances in Deep Learning-based Technological Applications*. Springer International Publishing, 2020, pp. 237–256.

[60] Y. LeCun, Y. Bengio, and G. Hinton, “Deep Learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[61] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, “Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification,” *Frontiers in Neuroscience*, vol. 11, p. 682, Dec. 2017.

[62] W. Fang, Y. Chen, J. Ding, Z. Yu, T. Masquelier, D. Chen, L. Huang, H. Zhou, G. Li, and Y. Tian, “SpikingJelly: An open-source machine learning infrastructure platform for spike-based intelligence,” *Science Advances*, vol. 9, no. 40, p. eadi1480, Oct. 2023.

[63] F. Zenke and S. Ganguli, “SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks,” *Neural Computation*, vol. 30, no. 6, 2018.

[64] J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, “Training Spiking Neural Networks Using Lessons From Deep Learning,” *CoRR*, vol. abs/2109.12894, 2021.

[65] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, S. J. Thorpe, and T. Masquelier, “Bio-inspired digit recognition using reward-

modulated spike-timing-dependent plasticity in deep convolutional networks," *Pattern Recognition*, vol. 94, pp. 87–95, Oct. 2019.

[66] 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, Jul. 2019.

[67] S. Aenugu, A. Sharma, S. Yelamarthy, H. Hazan, Philip.S.Thomas, and R. Kozma, "Reinforcement Learning with a Network of Spiking Agents," in *Real Neurons & Hidden Units: Future Directions at the Intersection of Neuroscience and Artificial Intelligence @ NeurIPS 2019*, Jul. 2022.

[68] S. Sharmin, P. Panda, S. S. Sarwar, C. Lee, W. Ponghiran, and K. Roy, "A Comprehensive Analysis on Adversarial Robustness of Spiking Neural Networks," in *International Joint Conference on Neural Networks*, 2019, pp. 1–8.

[69] N. Zheng and P. Mazumder, "Learning in Memristor Crossbar-Based Spiking Neural Networks Through Modulation of Weight-Dependent Spike-Timing-Dependent Plasticity," *IEEE Transactions on Nanotechnology*, vol. 17, no. 3, pp. 520–532, May 2018.

[70] S. Guo, "State and Action Abstraction in Reinforcement Learning," Ph.D. dissertation, Tsinghua University, May 2021.

[71] G. Liu, W. Deng, X. Xie, L. Huang, and H. Tang, "Human-Level Control Through Directly Trained Deep Spiking Q-Networks," *IEEE Transactions on Cybernetics*, pp. 1–12, 2022.

[72] G. Tang, N. Kumar, R. Yoo, and K. Michmizos, "Deep Reinforcement Learning with Population-Coded Spiking Neural Network for Continuous Control," in *Conference on Robot Learning*. PMLR, Oct. 2021, pp. 2016–2029.

[73] G. Tang, N. Kumar, and K. P. Michmizos, "Reinforcement co-Learning of Deep and Spiking Neural Networks for Energy-Efficient Mapless Navigation with Neuromorphic Hardware," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Oct. 2020, pp. 6090–6097.

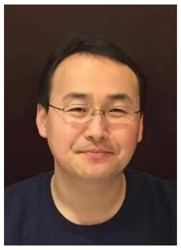
[74] D. Chen, P. Peng, T. Huang, and Y. Tian, "Deep Reinforcement Learning with Spiking Q-learning," *CoRR*, vol. abs/2201.09754, 2022.

Zhile Yang received the B.Eng. and M.S. degrees from the Department of Automation, Tsinghua University, Beijing, China, in 2018 and 2021, respectively. He is currently pursuing the Ph.D. degree with the School of Computing, University of Leeds, Leeds, UK. His current research interests include reinforcement learning and computational neuroscience.

Shangqi Guo received the B.S. in mathematics and physics basic science from the University of Electronic Science and Technology of China in 2015, and the Ph.D. degree from the Department of Automation, Tsinghua University in 2021. He is now a postdoctoral fellow, in the Department of Precision Instruments, Tsinghua University, China. His current research interests include reinforcement learning, spiking neural networks, and brain-inspired large language models.

Ying Fang received the B.S. degree in Automation from Xi'an Jiaotong University, Xi'an, China, in 2014, and the Ph.D. degree from the Department of Automation, Tsinghua University, Beijing, China, in 2020. She is currently an Associate Professor at Fujian Normal University. Research primarily focuses on the inference and learning of spiking neural networks (SNNs) and related machine learning algorithms.

Zhaofei Yu (Member, IEEE) received the B.S. degree from the Hong Shen Honors School, College of Optoelectronic Engineering, Chongqing University, Chongqing, China, in 2012 and the Ph.D. degree from the Automation Department, Tsinghua University, Beijing, China, in 2017. He is currently an Assistant Professor with the Institute for Artificial Intelligence, Peking University, Beijing. His current research interests include artificial intelligence, brain-inspired computing, and computational neuroscience.



Jian K. Liu received the Ph.D. degree in mathematics from University of California at Los Angeles, USA in 2009. He is currently an Associate Professor with School of Computer Science, Centre for Human Brain Health, University of Birmingham, UK. His research interests include fundamental questions of neural computation, computational neuroscience, and brain-inspired computation for intelligence, as well as applications to medicine.

Appendix A
Proof of Theorems

A.1 Proof of Theorem 1

Recall that the mean-field policy inference function is

$$q_i = \frac{1}{Z(q_{G(i)})} \exp\{\boldsymbol{w}_{\mathrm{row},i}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\mathrm{col},i}^{\mathrm{T}}\boldsymbol{q} + b_i\}, \tag{22}$$

where $Z(q_{G(i)}) = \sum_{j\in G(i)} \exp\{\boldsymbol{w}_{\mathrm{row},i}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\mathrm{col},i}^{\mathrm{T}}\boldsymbol{q} + b_i\}$, $i = 1, \ldots, (n_h d_h + d_a)$, $G(i)$ is the set of indices of the neurons in the same circuit as neuron i, and $\boldsymbol{w}_{\mathrm{row},i}$ and $\boldsymbol{w}_{\mathrm{col},i}$ are respectively the i-th row and column of matrix $\boldsymbol{W}$ (in the shape of a column vector), which corresponds to the synapses connected to neuron i. b_i is the i-th element in vector $\boldsymbol{b}$.

For each w_{jk}, There is

$$\begin{aligned}\frac{\partial q_i}{\partial w_{jk}} =& - Z^{-2}(q_{G(i)})\frac{\partial Z(q_{G(i)})}{\partial w_{jk}} \exp\left\{\boldsymbol{w}_{\mathrm{row},i}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\mathrm{col},i}^{\mathrm{T}}\boldsymbol{q} + b_i\right\} \\ &+ Z^{-1}(q_{G(i)}) \exp\left\{\boldsymbol{w}_{\mathrm{row},i}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\mathrm{col},i}^{\mathrm{T}}\boldsymbol{q} + b_i\right\} \cdot \sum_{m=1}^{N}\left[\frac{\partial(w_{im}+w_{mi})}{\partial w_{jk}} q_m + (w_{im}+w_{mi})\frac{\partial q_m}{\partial w_{jk}}\right] \\ =& - q_i Z^{-1}(q_{G(i)})\frac{\partial Z(q_{G(i)})}{\partial w_{jk}} + q_i\sum_{m=1}^{N}\left[\frac{\partial(w_{im}+w_{mi})}{\partial w_{jk}} q_m + (w_{im}+w_{mi})\frac{\partial q_m}{\partial w_{jk}}\right].\end{aligned} \tag{23}$$

For the term $\frac{\partial Z(q_{G(i)})}{\partial w_{jk}}$, there is

$$\begin{aligned}\frac{\partial Z(q_{G(i)})}{\partial w_{jk}} =& \frac{\partial}{\partial w_{jk}}\left\{\sum_{m\in G(i)} \exp\left\{\boldsymbol{w}_{\mathrm{row},m}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\mathrm{col},m}^{\mathrm{T}}\boldsymbol{q} + b_m\right\}\right\} \\ =& \sum_{m\in G(i)}\left\{\exp\left[\boldsymbol{w}_{\mathrm{row},m}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\mathrm{col},m}^{\mathrm{T}}\boldsymbol{q} + b_m\right] \cdot \sum_{n=1}^{N}\left[\frac{\partial(w_{mn}+w_{nm})}{\partial w_{jk}} q_n + (w_{mn}+w_{nm})\frac{\partial q_n}{\partial w_{jk}}\right]\right\}.\end{aligned} \tag{24}$$

So we have

$$\begin{aligned}\frac{\partial q_i}{\partial w_{jk}} =& - q_i\sum_{m\in G(i)}\left\{q_m \sum_{n=1}^{N}\left[\frac{\partial(w_{mn}+w_{nm})}{\partial w_{jk}} q_n + (w_{mn}+w_{nm})\frac{\partial q_n}{\partial w_{jk}}\right]\right\} \\ &+ q_i\sum_{n=1}^{N}\left[\frac{\partial(w_{in}+w_{ni})}{\partial w_{jk}} q_n + (w_{in}+w_{ni})\frac{\partial q_n}{\partial w_{jk}}\right].\end{aligned} \tag{25}$$

Similarly, for each b_j, there is

$$\frac{\partial q_i}{\partial b_j} = -q_i\sum_{m\in G(i)}\left\{q_m\left[\frac{\partial b_m}{\partial b_j} + \sum_{n=1}^{N}(w_{mn}+w_{nm})\frac{\partial q_n}{\partial_{b_j}}\right]\right\} + q_i\left[\frac{\partial b_i}{\partial b_j} + \sum_{n=1}^{N}(w_{in}+w_{ni})\frac{\partial q_n}{\partial_{b_j}}\right]. \tag{26}$$

By respectively arranging Eq. (25) and Eq. (26) for each q_i into vectors, and combining the terms into matrices, we can get the Eq. (15) in Theorem 1. □

A.2 Proof of Theorem 2

The condition is the same as that in the proof of Theorem 1. The approximate differentiation of firing rate q_i with respect to w_{jk} and b_j are:

$$\begin{aligned}\frac{\partial\log(q_i)}{\partial w_{jk}} =& \sum_{m=1}^{N}\left[\frac{\partial(w_{im}+w_{mi})}{\partial w_{jk}} q_m\right] - \frac{1}{Z(q_{G(i)})}\sum_{m\in G(i)}\left[\exp\{\boldsymbol{w}_{\mathrm{row},m}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\mathrm{col},m}^{\mathrm{T}}\boldsymbol{q} + b_m\} \cdot \sum_{n=1}^{N}\frac{\partial(w_{mn}+w_{nm})}{\partial w_{jk}} q_n\right] \\ =& \sum_{m=1}^{N}\left[\frac{\partial(w_{im}+w_{mi})}{\partial w_{jk}} q_m\right] - \sum_{m\in G(i)}\left[q_m \cdot \sum_{n=1}^{N}\frac{\partial(w_{mn}+w_{nm})}{\partial w_{jk}} q_n\right],\end{aligned} \tag{27}$$

$$\begin{aligned}\frac{\partial\log(q_i)}{\partial b_j} =& \frac{\partial b_i}{\partial b_j} - \frac{1}{Z(q_{G(i)})} \cdot \sum_{m\in G(i)}\left[\frac{\partial b_m}{\partial b_j} \cdot \exp\{\boldsymbol{w}_{\mathrm{row},m}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\mathrm{col},m}^{\mathrm{T}}\boldsymbol{q} + b_m\}\right] \\ =& \frac{\partial b_i}{\partial b_j} - \sum_{m\in G(i)}\left[q_m\frac{\partial b_m}{\partial b_j}\right].\end{aligned} \tag{28}$$

Similar to the proof of Theorem 1, by respectively arranging Eq. (27) and Eq. (28) for each q_i into vectors, and combining the terms on the right side into matrices, we can get the Eq. (16) in Theorem 2. □

Appendix B
SVPG Algorithm Details

In the main text, we provide the workflow of SVPG for the PPO base algorithm (see Algorithm 1). The workflow for the REINFORCE algorithm is presented in Algorithm 2. For conciseness, these algorithms do not cover the spike-based implementation, i.e., they are for the rate-based implementations.

The spike-based implementations can be achieved by replacing Eq.(11) with Eq.(13). For the simulation of the spiking process, we use the rectangle function Eq. (4) for the excitatory postsynaptic potential κ. In MNIST, we set the total length to 100, and set the length τ_3 of the window to 30. In GYMIP, we set them to 200 and 90. Due to the high computation cost in simulation and the complexity in the DOOM task, the spike-based SVPG is not tested in DOOM.

For the implementation of the critic network in Algorithm 1, we use an MLP to estimate the state values. For the MNIST and DOOM tasks, the MLP has three hidden layers with sizes $\langle 2048, 1024, 1024 \rangle$, and the output is the state value (with size 1). For the GYMIP task, the MLP has two hidden layers with sizes $\langle 64, 64 \rangle$ and the output is the state-action values (with size 5).

Algorithm 2 SVPG with REINFORCE

Input: Discount factor γ. Training episode number N_{epi}. Inference iteration number N_{iter}. Learning rate η.
Parameter: Network shape n_h, d_h, d_a, d_s.
Output: RWTA Network parameter θ.

```
1:  Initialize $\theta$ to zero.
2:  for Episode = 1, ..., $N_{\text{epi}}$ do
3:      Clear memory buffer $\mathcal{D}$.
4:      for Training step $t = 1, \ldots, T$ do
5:          Observe and encode state $s_t$.
6:          Random initialize $\boldsymbol{q}_a$ and $\boldsymbol{q}_h$.
7:          Iterate Eq.(11) until convergence. {Inference}
8:          Sample action $a_t$ using $\boldsymbol{q}_a$.
9:          Perform $a_t$, observe reward $r_t$ and new state $s_{t+1}$.
10:         Store $\langle s_t, a_t, r_t, s_{t+1}, \boldsymbol{q}, \boldsymbol{v} \rangle$ into $\mathcal{D}$.
11:     end for
12:     Get data from $\mathcal{D}$.
13:     Calculate gradient using Eq.(14), Eq.(17), Eq.(16). {Optimization}
14:     Update $\theta \leftarrow \theta + \eta \nabla \theta$ .
15: end for
```

Appendix C
Experiment Details

C.1 Task Details

MNIST. In this task, each episode contains only one time step. The agent's observation is randomly selected from the MNIST dataset. The action space contains 10 actions, each corresponding to the 10 classes. The input images are of size 28×28 and are in grayscale. The range of values is converted to $[0, 1]$ by dividing 255. For all the algorithms compared, the input images are stretched to vectors (so have length 784). A reward of $+1$ is given when the action is correct and -1 the opposite. The maximum length of training is set to 20k steps and each step samples a batch of 100 images.

GYMIP. Each episode has a maximum of 200 time steps, with a reward of $+1$ for each step. The episodes end early if the pendulum falls. The observation is a 4-dimensional vector with no predefined ranges. To normalize the observations to the range of $[0, 1]$, we use a random policy to sample from the environment and use the samples' range to determine a linear mapping to the range of $[0, 1]$. In our experiments, the sampled ranges are $[-0.4, 0.4]$, $[-0.2, 0.2]$, $[-1.7, 1.7]$, $[-1.25, 1.25]$. The action space consists of 5 discrete actions, evenly extracted from the range $[-3, 3]$. The maximum length of training is set to 2k episodes.

DOOM. In this task, the game lasts a maximum of 2100 screen frames. We adopt the frame-skipping technique, i.e., repeat an action for a fixed number of frames. In our setting, we choose to repeat each action for 4 frames, resulting in a maximum number of 525 time steps in each episode. The rewards are determined by the player states, i.e., -50 when dead, $+10$ when picking up a health kit, and $+1$ otherwise. The game screen has a resolution of 320×240, which is transformed into grayscale, resized to 80×60, and reshaped to a vector with length 4800 to serve as the state observation. The action space contains 5 actions corresponding to the keyboard actions in the game: "move forward", "turn left", "turn right", "turn left while moving forward", and "turn right while moving forward". The maximum length of training is set to 2k episodes for SVPG and BP, and 10k episodes for BPTT since it exhibits slower learning.

AI2THOR. In this task, each episode contains a maximum of 200 steps. The target is to find a television in the room. The agent needs to be within 1.5m distance from the television and the television needs to be in the agent's view to mark a successful navigation. The episode ends early if the target object is found. The room map is `FloorPlan_Train7_5` from the AI2THOR (specifically, RoboTHOR) platform [36]. The starting points are randomly selected from 38 randomly generated points (30 for training, 8 for validation/testing). The state observation is a 80 $\times$ 60 RGB image obtained from the agent's viewpoint. The parameters of the camera, e.g., angle of view, are the default ones provided by the AI2THOR platform. We convert the observed image to grayscale and reshape it to a 4800-length vector. Figure 18 presents an example of the original RGB image and the state observation after pre-processing. The action space consists of 5 actions: forward, turn left, turn right, move left, and move right. These actions are implemented by a list of operations provided by the AI2THOR platform. For example, turning left is achieved by `RotateLeft` and `MoveAhead`; moving left is achieved by `RotateLeft`, `MoveAhead`, and then `RotateRight`. The step sizes are 0.15m for movements and 90 degrees for rotations. The reward is defined to be: +50 for target-reaching, -5 for failure in action (e.g., collision), +1 for target-approaching, and -1 for target-deviation. The maximum length of training is set to 8k episodes.

Fig. 18. Example state observations in the AI2THOR task. (a) An original state observation. (b) A pre-processed state observation.

ROBOTARM. In this task, the robot arm is tasked to reach a cude on the table. The simulation environment is built upon the example panda control scene from PyRep [57]. We added a vision sensor and a cube, and configured the initial pose of the robot arm. The scene is included in the code repository https://github.com/yzlc080733/SVPG2023/tree/main/ROBOTARM/env_data. Each episode contains a maximum of 60 steps. The episode ends early if the target is reached. The state is a 64 $\times$ 64 RGB image obtained from a vision sensor attached at the end point of the arm. For pre-processing, we first convert the image to grayscale, then clipped the pixel values to the range of $[0.4, 0.9]$ and finally linearly map it to the range of $[0, 1]$. Figure 19 presents an overview of the environment and an example of the original RGB image and the image after pre-processing. The input to the networks is a length-4096 vector reshaped from the processed image. The action space consists of 5 actions: move in 4 ways horizontally with a step size of 0.03m, and move upward/downward with a step size of 0.05m. The end-effector of the arm is limited to move in a 0.2m $\times$ 0.3m $\times$ 0.22m rectangular space. The target object is a cube with a side length of 0.05m, and its position is randomly selected from 50 randomly generated positions (among which 45 for training and 5 for validation/testing). The reward is defined to be: +10 for target-reaching, +1 for target-approaching, and 0 otherwise. The maximum length of training is set to 10k episodes.

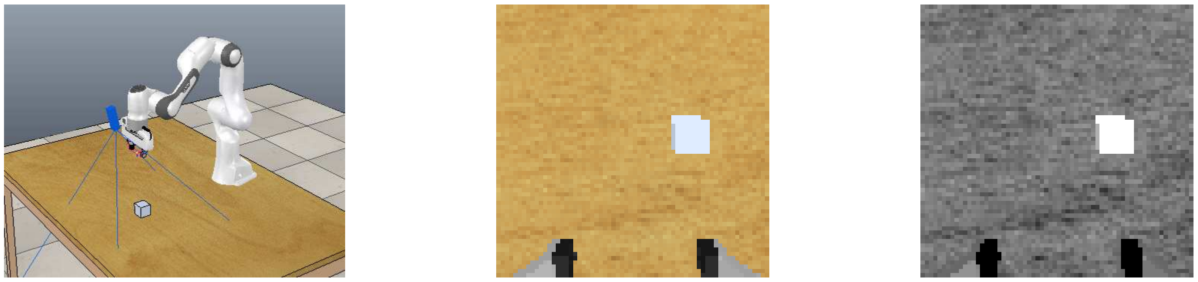


Fig. 19. Example images in the ROBOTARM task. (a) An overview of the environment. The blue cuboid close to the gripper is the vision sensor. (b) An example state observation. (c) A pre-processed state observation.

C.2 Implementation Details

RL hyper-parameters. For all the tasks in our experiments, we set the discount factor γ to be 0.97. This makes the discounted return reflect the length of the episodes in GYMIP and DOOM, so that the agent learns to complete the task. In MNIST, DOOM, AI2THOR, and ROBOTARM, we use Monte-Carlo sampling to learn the critic; in GYMIP, we use temporal difference to learn the critic and adopt a memory buffer with size 1000. We set the epoch number in PPO to 5, and set the clipping parameter to 0.2. Due to the instability in training BPTT on DOOM, we specially set the epoch number to 10 for BPTT on DOOM. To encourage the agent to explore different actions, we use a weighted sum of the policy gradient and the entropy of the agents' policy distribution to train the agent. This introduces a hyper-parameter of the entropy ratio.

In practice, we find that the learning rate and the entropy ratio have a large impact on the zero-noise test results. Therefore, we tune these two hyper-parameters for each method and report the one with the best testing performance. On MNIST and GYMIP, we compare entropy ratio with values $\{0, 0.1, 0.2, 0.5, 1, 2, 5, 10\}$ and learning rate with values $\{0.01, 0.001, 0.0001\}$. On DOOM, we compare entropy ratio with values $\{0.02, 0.2, 2, 5\}$ and learning rates $\{0.001, 0.0001\}$. On AI2THOR, we compare the entropy ratio with values $\{0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5\}$ and learning rates $\{0.0001, 0.00001\}$. On ROBOTHOR, we compare the entropy ratio with values $\{0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10\}$ and learning rates $\{0.0001, 0.00001\}$. For each setting of the hyper-parameters, we perform 10 independent trainings for GYMIP and DOOM, and 3 independent trainings for MNIST, AI2THOR, and ROBOTARM. We provide the tuning results in Figure 20, 21, and 22. The values are the mean values of the zero-perturbation testing performances and the bold values indicate the settings that are used in robustness comparisons.

As mentioned in the main text, we select the best hyper-parameter for each method on each task to compare their robustness. On MNIST, DOOM, and ROBOTARM, this is done by comparing their mean zero-noise performances; when the performances are the same, the performance at a level of perturbation is considered. For input perturbations, the level value is set to 0.2; for network parameter perturbations, the level value is 2.0. On GYMIP, we consider the average of all hyper-parameters that generate the best zero-noise performance; to tolerate noises in the zero-noise performance, we adopt the hyper-parameters with zero-noise performance greater than or equal to 99% the best performance. On AI2THOR, BP and ANN2SNN generate multiple best zero-noise results; all the corresponding hyper-parameters are considered.

SVPG		Entropy Ratio							
		0	0.1	0.2	0.5	1	2	5	10
Learning Rate	0.01	0.7448	0.8056	0.8042	0.8914	0.8605	0.8894	0.4123	0.3054
	0.001	0.9284	0.9270	**0.9292**	0.9271	0.9270	0.9262	0.8467	0.7938
	0.0001	0.9233	0.9236	0.9232	0.9230	0.9235	0.9231	0.9088	0.8997

(a)

SVPG-shrink		Entropy Ratio							
		0	0.1	0.2	0.5	1	2	5	10
Learning Rate	0.01	0.8053	0.8344	0.8960	0.9211	0.8598	0.9139	0.4226	0.3295
	0.001	0.9267	0.9281	**0.9282**	0.9269	0.9261	0.9242	0.8491	0.8018
	0.0001	0.9235	0.9246	0.9237	0.9243	0.9252	0.9232	0.9157	0.9036

(b)

BP		Entropy Ratio							
		0	0.1	0.2	0.5	1	2	5	10
Learning Rate	0.01	0.5871	0.9339	0.9357	0.8858	0.5548	0.2746	0.1121	0.1093
	0.001	0.9781	0.9763	0.9691	0.9565	0.9512	0.9276	0.9069	0.8924
	0.0001	0.9760	0.9761	0.9774	0.9779	**0.9792**	0.9739	0.9639	0.9578

(c)

BPTT		Entropy Ratio							
		0	0.1	0.2	0.5	1	2	5	10
Learning Rate	0.01	0.4356	0.4257	0.5191	0.1884	0.1032	0.1032	0.1695	0.1370
	0.001	0.7783	0.8371	0.8580	0.9001	0.9252	0.4060	0.2116	0.1638
	0.0001	0.9729	0.9735	0.9744	**0.9748**	0.9745	0.9624	0.9434	0.9211

(d)

ANN2SNN		Entropy Ratio							
		0	0.1	0.2	0.5	1	2	5	10
Learning Rate	0.01	0.5866	0.9349	0.9292	0.8805	0.5477	0.2680	0.1121	0.1093
	0.001	0.9788	0.9764	0.9660	0.9525	0.9490	0.9253	0.9040	0.8899
	0.0001	0.9765	0.9766	0.9779	0.9782	**0.9784**	0.9727	0.9618	0.9550

(e)

Fig. 20. Hyper-parameter tuning on MNIST.

During training, a checkpoint of the network parameters is saved every 100 episodes. The checkpoints are validated using the validation environment and the best one is used in testing. The validation environment in the MNIST task is created by randomly dividing the training set according to a ratio of 9:1; the latter part is used as the validation set. The validation environments in GYMIP and DOOM are the same as the training environments. For AI2THOR, the validation environment uses the same scene as in training, but the starting points are randomly sampled from a list of positions different from training. For ROBOTARM, in the validation environment, the initial position of the target cube is randomly sampled from a list different from training.

SVPG		Entropy Ratio							
		0	0.1	0.2	0.5	1	2	5	10
Learning Rate	0.01	96.58	149.96	112.58	173.95	184.05	**200.00**	183.24	165.09
	0.001	166.17	143.15	197.29	182.36	**200.00**	**200.00**	**200.00**	**200.00**
	0.0001	62.56	88.81	132.89	149.40	148.90	161.22	140.45	137.65

(a)

BP		Entropy Ratio							
		0	0.1	0.2	0.5	1	2	5	10
Learning Rate	0.01	105.02	**200.00**	**200.00**	**200.00**	**200.00**	**200.00**	**198.99**	183.77
	0.001	99.71	**198.52**	**200.00**	**200.00**	**200.00**	**200.00**	**199.56**	191.98
	0.0001	**199.97**	**200.00**	**200.00**	**200.00**	**200.00**	**200.00**	**199.79**	146.59

(b)

BPTT		Entropy Ratio							
		0	0.1	0.2	0.5	1	2	5	10
Learning Rate	0.01	128.25	128.14	92.38	119.06	121.89	139.57	165.00	103.20
	0.001	178.22	137.88	141.44	159.98	143.33	174.51	167.23	120.84
	0.0001	141.44	176.86	129.18	191.03	159.71	192.64	**198.18**	113.56

(c)

ANN2SNN		Entropy Ratio							
		0	0.1	0.2	0.5	1	2	5	10
Learning Rate	0.01	104.45	188.40	187.12	172.05	189.72	**199.83**	181.75	150.33
	0.001	99.70	176.54	175.77	191.43	189.83	197.75	171.04	157.82
	0.0001	187.74	190.20	**200.00**	187.35	**198.34**	179.52	167.63	99.39

(d)

Fig. 21. Hyper-parameter tuning on GYMIP.

SVPG		Entropy Ratio			
		0.02	0.2	2	5
Learning Rate	0.001	503.24	**525.00**	**525.00**	462.67
	0.0001	435.57	513.73	517.78	483.61

(a)

BP		Entropy Ratio			
		0.02	0.2	2	5
Learning Rate	0.001	**525.00**	224.25	129.12	124.32
	0.0001	**525.00**	517.14	292.26	217.13

(b)

BPTT		Entropy Ratio			
		0.02	0.2	2	5
Learning Rate	0.001	121.92	124.91	108.34	97.14
	0.0001	**394.98**	296.02	112.10	127.80

(c)

ANN2SNN		Entropy Ratio			
		0.02	0.2	2	5
Learning Rate	0.001	**525.00**	200.85	125.76	122.96
	0.0001	475.15	495.38	283.69	223.23

(d)

Fig. 22. Hyper-parameter tuning on DOOM.

Network Sizes.

- For the MNIST task, SVPG uses an RWTA network with 784 input neurons, 50 hidden WTA circuits each with 10 neurons, and 10 output neurons (527850 learnable parameters). BP, BPTT, and ANN2SNN use layered networks with 784 input neurons, 1 hidden layer with 500 neurons, and 10 output neurons (397510 learnable hyperparameters). SVPG-shrink uses a smaller RWTA network where the number of hidden WTA circuits is changed to 39 so that the total number of learnable parameters is close to other methods (392000 learnable parameters).
- For the GYMIP task, SVPG uses an RWTA network with 4 input neurons, 8 hidden WTA circuits each with 8 neurons, and 5 output neurons. BP, BPTT, and ANN2SNN use layered networks with 4 input neurons, 1 hidden layer with 64 hidden neurons, and 5 output neurons.
- For the DOOM and the AI2THOR task, SVPG uses an RWTA network with 4800 input neurons, 50 hidden WTA circuits each with 10 neurons, and 5 output neurons. BP, BPTT, and ANN2SNN use layered networks with 4800 input neurons, 1 hidden layer with 500 hidden neurons, and 5 output neurons.
- For the ROBOTARM task, SVPG uses an RWTA network with 4096 input neurons, 50 hidden WTA circuits each with 10 neurons, and 5 output neurons. BP, BPTT, and ANN2SNN use layered networks with 4096 input neurons, 1 hidden layer with 500 hidden neurons, and 5 output neurons.

Optimizer. For the MNIST, DOOM, AI2THOR, and ROBOTARM task, we use the Adam optimizer with hyper-

SVPG		Entropy Ratio							
		0.02	0.05	0.1	0.2	0.5	1	2	5
Learning	0.0001	200.00	142.13	142.15	200.00	200.00	146.04	83.04	200.00
Rate	0.00001	**24.00**	82.67	82.67	44.83	27.67	33.67	37.38	32.38

(a)

BP		Entropy Ratio							
		0.02	0.05	0.1	0.2	0.5	1	2	5
Learning	0.0001	34.67	28.60	40.13	28.77	**24.00**	82.69	200.00	141.79
Rate	0.00001	38.42	32.81	25.63	26.33	**24.00**	24.02	24.04	24.06

(b)

BPTT		Entropy Ratio							
		0.02	0.05	0.1	0.2	0.5	1	2	5
Learning	0.0001	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00
Rate	0.00001	200.00	200.00	**82.77**	141.33	200.00	200.00	141.33	141.73

(c)

ANN2SNN		Entropy Ratio							
		0.02	0.05	0.1	0.2	0.5	1	2	5
Learning	0.0001	87.42	66.73	84.79	45.98	24.02	83.65	200.00	141.63
Rate	0.00001	141.58	40.90	32.69	62.21	**24.00**	24.13	24.15	**24.00**

(d)

Fig. 23. Hyper-parameter tuning on AI2THOR.

SVPG		Entropy Ratio									
		0.01	0.02	0.05	0.1	0.2	0.5	1	2	5	10
Learning	0.0001	31.50	24.87	35.20	41.93	38.40	38.33	45.47	12.07	10.47	19.83
Rate	0.00001	16.60	10.93	8.47	11.47	10.17	8.20	**7.43**	12.90	9.07	7.47

(a)

BP		Entropy Ratio									
		0.01	0.02	0.05	0.1	0.2	0.5	1	2	5	10
Learning	0.0001	32.20	42.47	60.00	52.87	46.00	56.60	60.00	60.00	60.00	60.00
Rate	0.00001	7.00	7.53	7.33	7.13	7.13	7.13	**6.93**	7.40	7.13	7.67

(b)

BPTT		Entropy Ratio									
		0.01	0.02	0.05	0.1	0.2	0.5	1	2	5	10
Learning	0.0001	60.00	60.00	60.00	60.00	60.00	60.00	60.00	60.00	60.00	60.00
Rate	0.00001	60.00	27.87	31.53	52.73	**25.40**	42.47	31.80	60.00	60.00	60.00

(c)

ANN2SNN		Entropy Ratio									
		0.01	0.02	0.05	0.1	0.2	0.5	1	2	5	10
Learning	0.0001	39.07	42.47	60.00	52.87	46.40	58.07	60.00	60.00	60.00	60.00
Rate	0.00001	27.40	20.53	16.93	20.20	20.67	23.93	17.13	**7.20**	10.27	14.00

(d)

Fig. 24. Hyper-parameter tuning on ROBOTARM.

parameters set as $\epsilon = 1 \times 10^{-8}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. For the GYMIP task, we use the RMSprop optimizer with $\epsilon = 1 \times 10^{-8}$, $\alpha = 0.99$. For the critic network in all three tasks, we use the Adam optimizer with $\epsilon = 1 \times 10^{-8}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$.

Appendix D
Additional Experiment Results

D.1 Additional Comparison of Rate-Based SVPG and Spike-Based SVPG

In the main text, we present a part of the results of the comparison of the spike-based and rate-based SVPG implementations. The complete set of comparison results is presented in Figure 25 and Figure 26, including more types of perturbations of input and network parameters. These results further support the analysis in the main text that the two implementations generate similar performances.

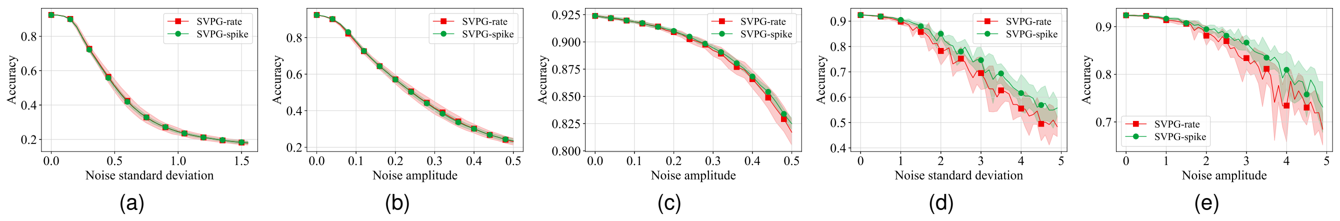


Fig. 25. Additional comparison of spike-based and rate-based SVPG implementations on MNIST. (a) Input Gaussian noise. (b) Input salt noise. (c) Input pepper noise. (d) Network Gaussian noise. (e) Network uniform noise.

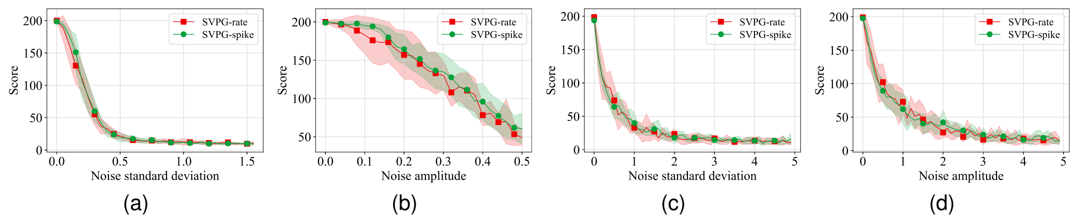


Fig. 26. Additional comparison of spike-based and rate-based SVPG implementations on GYMIP. (a) Input Gaussian noise. (b) Input uniform noise. (c) Network Gaussian noise. (d) Network uniform noise.

D.2 Additional Robustness Results

A representative part of the results of robustness tests are presented in the main text. Here is the complete set of results.

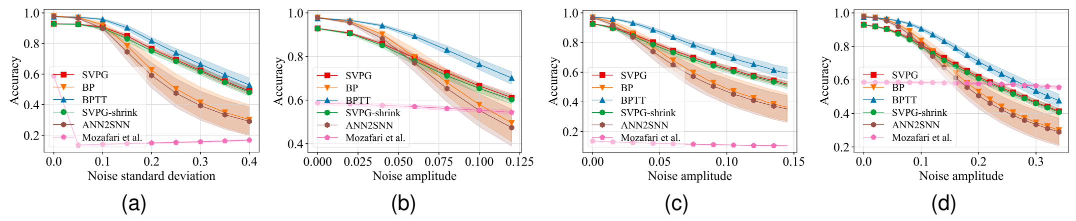


Fig. 27. MNIST. Input noises. (a) Gaussian. (b) Salt. (c) Gaussian&Salt. (d) Salt&Pepper.

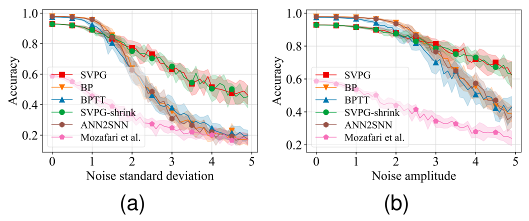


Fig. 28. MNIST. Network parameter noises. (a) Gaussian. (b) Uniform.

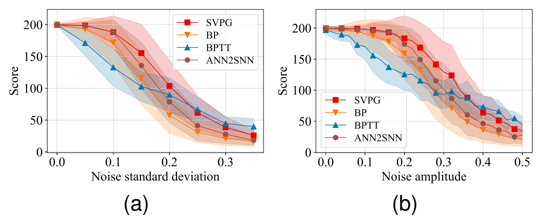


Fig. 29. GYMIP. Input noises. (a) Gaussian. (b) Uniform.

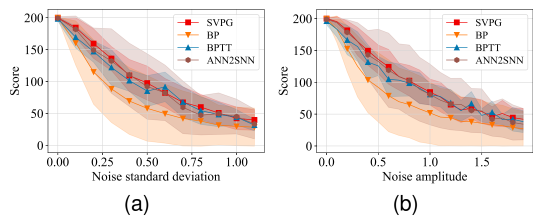


Fig. 30. GYMIP. Network parameter noises. (a) Gaussian. (b) Uniform.

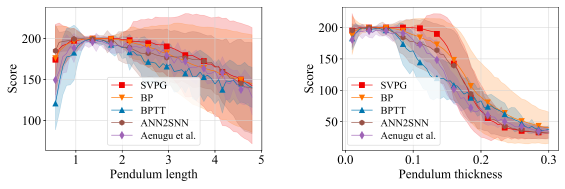

(a) (b)

Fig. 31. GYMIP. Environmental variations. (a) Pendulum length. (b) Pendulum thickness.

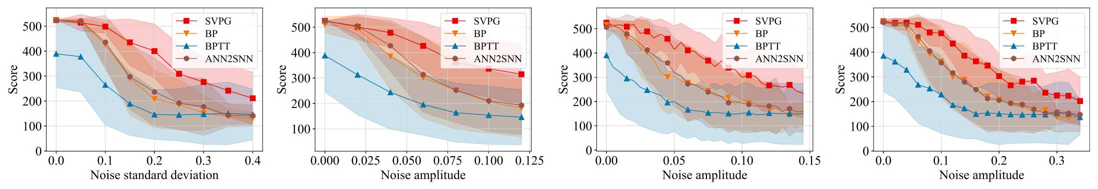

(a) (b) (c) (d)

Fig. 32. DOOM. Input noises. (a) Gaussian. (b) Salt. (c) Gaussian & salt. (d) Salt & pepper.

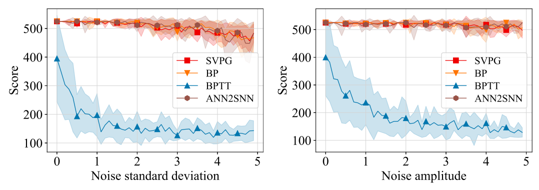

(a) (b)

Fig. 33. DOOM. Network parameter noises. (a) Gaussian. (b) Uniform.

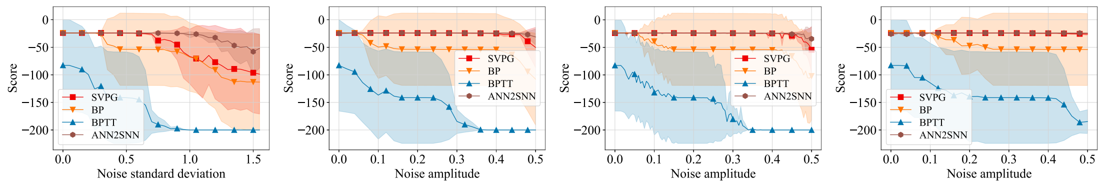

(a) (b) (c) (d)

Fig. 34. AI2THOR. Input noises. (a) Gaussian. (b) Salt. (c) Gaussian & salt. (d) Salt & pepper.

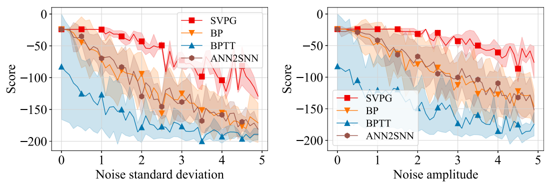

(a) (b)

Fig. 35. AI2THOR. Network parameter noises. (a) Gaussian. (b) Uniform.

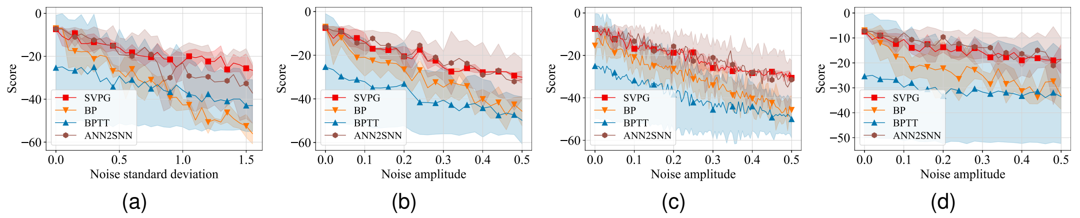


Fig. 36. ROBOTARM. Input noises. (a) Gaussian. (b) Salt. (c) Gaussian & salt. (d) Salt & pepper.

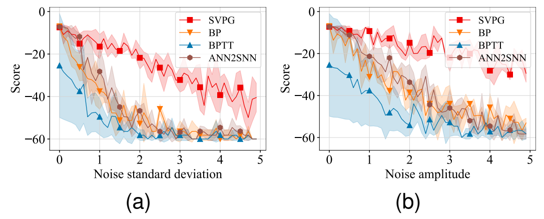


Fig. 37. ROBOTARM. Network parameter noises. (a) Gaussian. (b) Uniform.

D.3 Learning Curves

To show the sample efficiency of the methods, we draw the learning curves in each task. For each method and each task, a hyper-parameter (entropy ratio and learning rate) that produces the best testing performance is chosen. The results are presented in Figure 38. The horizontal axis is the training step number. The vertical axis is the validation score/performance (higher is better), which is measured during training in periods of 100 training steps. The curves are smoothed using exponential smoothing with smoothing factor $\alpha = 0.4$ ($0 < \alpha < 1$; a smaller α means stronger smoothing). For MNIST, GYMIP, and DOOM, the curves are the average of 10 independent trainings. For AI2THOR and ROBOTARM, the curves are the average of 3 independent trainings. The shades present the standard deviation values.

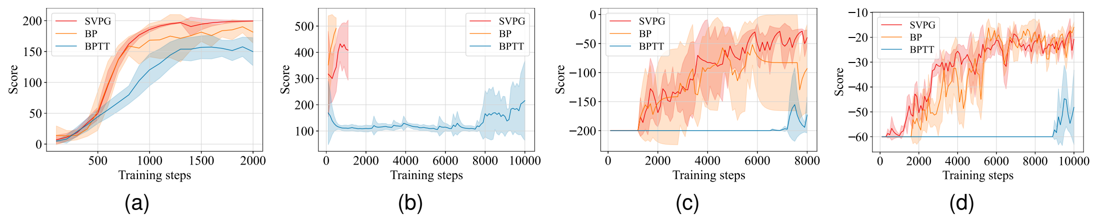


Fig. 38. Learning curves. (a) GYMIP. (b) DOOM. (c) AI2THOR. (d) ROBOTARM.

As shown, the curves of SVPG are generally close to those of BP. Both SVPG and BP learn steadily and faster than BPTT. This indicates that the sample efficiency of SVPG is similar to BP. SVPG has the potential to be applied to scenarios where the efficiency of a traditional ANN is acceptable.

D.4 Additional Visualizations

In the main text, we provide visualizations of the entropy values and the spike trains of WTA circuits on the MNIST task. Here we provide the results on the GYMIP task. Since GYMIP involves a sequence of decisions, it enables an investigation of the network dynamics in an updating environment.

We consider an untrained network and a trained network. For each network, we perform one test episode and record the network dynamics at each step. The results are presented respectively in Figure 39 and Figure 40. In the figures, the first column is the step number in the test episode. The second column is a visualization of the environment state. The third column presents the entropy values of the action WTA circuit and some hidden WTA circuits during the firing process. The last two columns respectively present the actual spike trains generated by neurons in a hidden WTA circuit and the action WTA circuit.

As shown, there is a significant difference between a random network and a trained network. In a trained network, the entropy values of WTA circuits quickly decrease in the firing process. Besides, the firing patterns of the presented hidden and action neurons change when the environment state changes. We also notice that the action WTA circuit does not always converge to output one action quickly; instead, it may gradually change its output to the final action, as shown in step 9 and step 50 in Figure 40. This indicates that the firing process is useful for the generation of the final decision output.

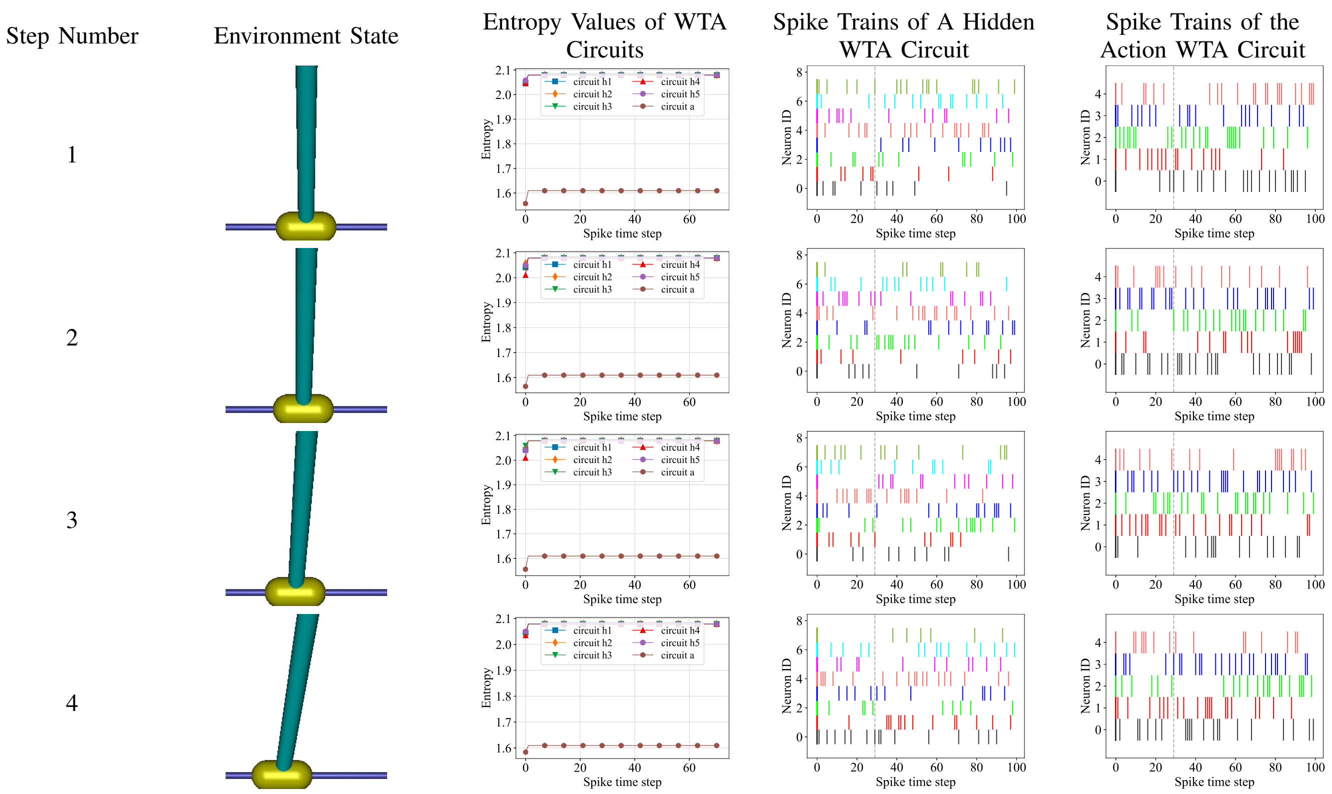


Fig. 39. Visualization of network dynamics of SVPG on GYMIP. Before training.

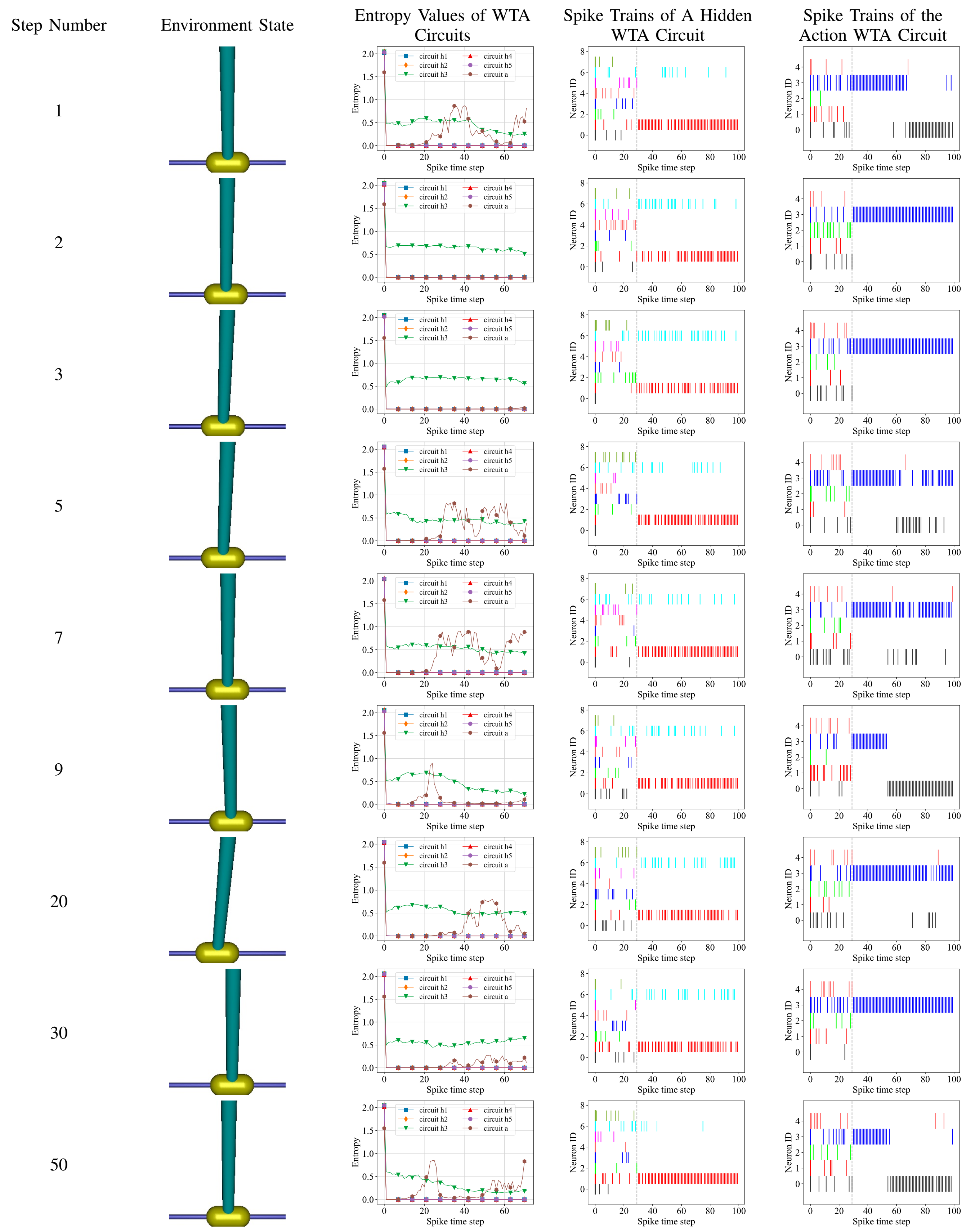


Fig. 40. Visualization of network dynamics of SVPG on GYMIP. After training.