A Proof of Theorems

A.1 Proof of Theorem 1

Recall that the mean-field inference function is

$$q_i = \frac{1}{Z(q_{G(i)})} \exp\{\boldsymbol{w}_{\text{row},i}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\text{col},i}^{\mathrm{T}}\boldsymbol{q} + b_i\}, \tag{16}$$

where $Z(q_{G(i)}) = \sum_{j\in G(i)} \exp\{\boldsymbol{w}_{\text{row},i}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\text{col},i}^{\mathrm{T}}\boldsymbol{q} + b_i\}$, $i = 1, \ldots, (n_h d_h + d_a)$, $G(i)$ is the set of indices of the neurons in the same circuit as neuron i, and $\boldsymbol{w}_{\text{row},i}$ 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 $\boldsymbol{b}$.

For each w_{jk}, There is

$$\begin{aligned}
\frac{\partial q_i}{\partial w_{jk}} =& -Z^{-2}(q_{G(i)}) \frac{\partial Z(q_{G(i)})}{\partial w_{jk}} \exp\left\{\boldsymbol{w}_{\text{row},i}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\text{col},i}^{\mathrm{T}}\boldsymbol{q} + b_i\right\} \\
&+ Z^{-1}(q_{G(i)}) \exp\left\{\boldsymbol{w}_{\text{row},i}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\text{col},i}^{\mathrm{T}}\boldsymbol{q} + b_i\right\} \\
&\cdot \sum_{m=1}^{N} \left[\frac{\partial (w_{im} + w_{mi})}{\partial w_{jk}} q_m + (w_{im} + w_{mi}) \frac{\partial q_m}{\partial w_{jk}} \right] \\
=& -q_i Z^{-1}(q_{G(i)}) \frac{\partial Z(q_{G(i)})}{\partial w_{jk}} + q_i \sum_{m=1}^{N} \left[\frac{\partial (w_{im} + w_{mi})}{\partial w_{jk}} q_m + (w_{im} + w_{mi}) \frac{\partial q_m}{\partial w_{jk}} \right].
\end{aligned} \tag{17}$$

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

$$\begin{aligned}
\frac{\partial Z(q_{G(i)})}{\partial w_{jk}} =& \frac{\partial}{\partial w_{jk}} \left\{ \sum_{m\in G(i)} \exp\left\{\boldsymbol{w}_{\text{row},m}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\text{col},m}^{\mathrm{T}}\boldsymbol{q} + b_m\right\} \right\} \\
=& \sum_{m\in G(i)} \left\{ \exp\left[\boldsymbol{w}_{\text{row},m}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\text{col},m}^{\mathrm{T}}\boldsymbol{q} + b_m\right] \right. \\
&\left. \cdot \sum_{n=1}^{N} \left[\frac{\partial (w_{mn} + w_{nm})}{\partial w_{jk}} q_n + (w_{mn} + w_{nm}) \frac{\partial q_n}{\partial w_{jk}} \right] \right\}.
\end{aligned} \tag{18}$$

So we have

$$\begin{aligned}
\frac{\partial q_i}{\partial w_{jk}} =& -q_i \sum_{m\in G(i)} \left\{ q_m \sum_{n=1}^{N} \left[\frac{\partial (w_{mn} + w_{nm})}{\partial w_{jk}} q_n + (w_{mn} + w_{nm}) \frac{\partial q_n}{\partial w_{jk}} \right] \right\} \\
&+ q_i \sum_{n=1}^{N} \left[\frac{\partial (w_{in} + w_{ni})}{\partial w_{jk}} q_n + (w_{in} + w_{ni}) \frac{\partial q_n}{\partial w_{jk}} \right].
\end{aligned} \tag{19}$$

Similarly, for each b_j, there is

$$\frac{\partial q_i}{\partial b_j} = -q_i \sum_{m\in G(i)} \left\{ q_m \left[\frac{\partial b_m}{\partial b_j} + \sum_{n=1}^{N} (w_{mn} + w_{nm}) \frac{\partial q_n}{\partial_{b_j}} \right] \right\} + q_i \left[\frac{\partial b_i}{\partial b_j} + \sum_{n=1}^{N} (w_{in} + w_{ni}) \frac{\partial q_n}{\partial_{b_j}} \right]. \tag{20}$$

By respectively arranging Eq. (19) and Eq. (20) for each q_i into vectors, and combining the terms into matrices, we can get the Eq. (10) 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:

$$\begin{aligned}\frac{\partial \log(q_i)}{\partial w_{jk}} =& \sum_{m=1}^{N}\left[\frac{\partial(w_{im}+w_{mi})}{\partial w_{jk}} q_m\right] \\ & - \frac{1}{Z(q_{G(i)})} \sum_{m\in G(i)} \left[\exp\{\boldsymbol{w}_{\text{row},m}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\text{col},m}^{\mathrm{T}}\boldsymbol{q} + b_m\} \cdot \sum_{n=1}^{N} \frac{\partial(w_{mn}+w_{nm})}{\partial w_{jk}} q_n\right] \\ =& \sum_{m=1}^{N}\left[\frac{\partial(w_{im}+w_{mi})}{\partial w_{jk}} q_m\right] - \sum_{m\in G(i)}\left[q_m \cdot \sum_{n=1}^{N} \frac{\partial(w_{mn}+w_{nm})}{\partial w_{jk}} q_n\right],\end{aligned} \tag{21}$$

$$\begin{aligned}\frac{\partial \log(q_i)}{\partial b_j} =& \frac{\partial b_i}{\partial b_j} - \frac{1}{Z(q_{G(i)})} \cdot \sum_{m\in G(i)}\left[\frac{\partial b_m}{\partial b_j} \cdot \exp\{\boldsymbol{w}_{\text{row},m}^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_{\text{col},m}^{\mathrm{T}}\boldsymbol{q} + b_m\}\right] \\ =& \frac{\partial b_i}{\partial b_j} - \sum_{m\in G(i)}\left[q_m \frac{\partial b_m}{\partial b_j}\right].\end{aligned} \tag{22}$$

Similar to the proof of Theorem 1, by respectively arranging Eq. (21) and Eq. (22) for each q_i into vectors, and combining the terms on the right hand side into matrices, we can get the Eq. (11) in Theorem 2. □

B SVPG Algorithm Details

We summarize the overall working flow with our RWTA network as Algorithm 1. For conciseness, this algorithm does not cover the implementation of spike-based inference and STDP-based optimization. Note that these spike-related implementations can be done by simply replacing Eq.(7) with Eq.(9), and applying Eq.(13).

In practice, Actor-Critic (AC) based methods are more commonly used than vanilla Policy Gradient (PG). The main difference between AC and PG is that AC learns an extra value function in the place of r_t in PG. To adapt our algorithm to AC, we add a multi-layer perceptron trained with backpropagation to estimate the critic value function, and use the estimated value to replace the r_t in Eq.(1).

The SVPG implemented with REINFORCE algorithm is presented in Algorithm 1. The equation numbers correspond to the equations in the main text.

Algorithm 1 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.

Initialize θ.
for Episode = 1, ..., N_{epi} **do**
  Clear memory buffer $\mathcal{D}$.
  for Training step t = 1, ..., T **do**
    Observe and encode state s_t.
    Random initialize $\boldsymbol{q}_a$ and $\boldsymbol{q}_h$.
    Iterate Eq.(7) until convergence. {Inference}
    Sample action a_t using $\boldsymbol{q}_a$.
    Perform a_t, observe reward r_t and new state s_{t+1}.
    Store $\langle s_t, a_t, r_t, s_{t+1}, \boldsymbol{q}, \boldsymbol{v} \rangle$ into $\mathcal{D}$.
  end for
  Get data from $\mathcal{D}$.
  Calculate gradient using Eq.(1), Eq.(12), Eq.(11). {Optimize}
  Update $\theta \leftarrow \theta + \eta \nabla \theta$
end for

C Experiment Details

C.1 Task Details

MNIST task. In this task, each episode includes only one time step. At the time step, the agent observes a randomly selected image from the MNIST training dataset, and selects one of the ten categories as its action. 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 (with length 784). A reward of $+1$ indicates a correct selection, -1 the opposite. The maximum length of training is set to 50k steps (in each step a randomly selected 100-size batch is used for training) so that in practice all the methods converge. During training, a checkpoint of the network parameters is saved every 100 training steps. After training, the checkpoint with the best accuracy in the testing set (without noise) is reloaded to do the testing. The MNIST testing dataset is used in testing.

GymIP task. Each episode has a maximum of 200 time steps, with a reward of $+1$ for each step. The episodes end early if the pendulum (pole) 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 20k episodes. During training, a checkpoint of the network parameters is saved every 20 episodes. After training, the checkpoint with the best performance is reloaded to do the testing.

C.2 Implementation Details

Network Sizes. 1) For the MNIST task, SVPG uses an RWTA network with 784 input neurons, 20 hidden WTA circuits each with 10 neurons, and 10 output neurons; SVPG-shrink uses a smaller RWTA network where the number of hidden WTA circuits is changed to 17 so that the total number of learnable parameters is close to other methods; BP, BPTT, ANN2SNN, and EP use layered networks with 784 input neurons, 1 hidden layer with 200 neurons, and 10 output neurons. 2) 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; SVPG-shrink uses a smaller RWTA network where the number of hidden WTA circuits is changed to 3. BP, BPTT, and ANN2SNN use layered networks with 4 input neurons, 1 hidden layer with 64 hidden neurons, and 5 output neurons. 3) For the GymIP task there is a critic network used in all the methods. This critic network is layered, with 4 input neurons, 2 hidden layers each with 64 neurons, and 5 output neurons.

Optimizer. For the compared methods, i.e., BP, BPTT, and EP, we select the stochastic gradient descent (SGD) with zero momentum for the MNIST task, and select the RMSprop optimizer for the GymIP task. For the SGD optimizer, we incompletely tried learning rates ranging from 0.001 to 0.3, and found 0.1 to be a good balance between training speed and stability. For the RMSprop optimizer, we use a learning rate of 0.001. As for SVPG, we use a learning rate of 0.1 in the MNIST task and 0.001 in the GymIP task.

RL hyper-parameters. The GymIP task involves sequential decisions which may have long-term effects on rewards. We select the discount rate γ to be 0.999 (close to 1) so that the discounted sum of rewards reflects the length of the episodes, maximizing which is the task objective. To stabilize training, we use a replay buffer with a size of 100. To encourage

the agent to explore more actions, we add an intrinsic exploration reward to the environment reward; the reward is calculated as the entropy of the agent's action distribution; the ratio of the environment reward and the intrinsic reward is 2:1.

D Additional Experiment Results

D.1 MNIST Input Noise Illustration

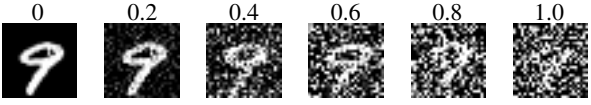
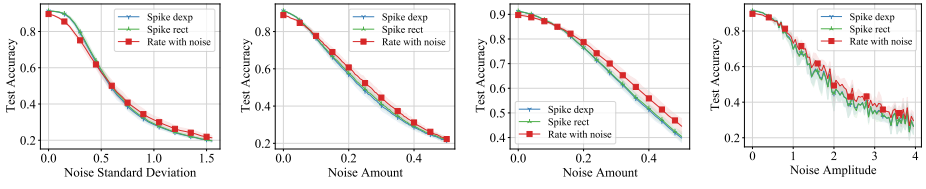


Figure 7: Some input images with different strengths of Gaussian noises in MNIST task. Standard deviation noted above images.

D.2 Additional Comparison of Three SVPG Implementations

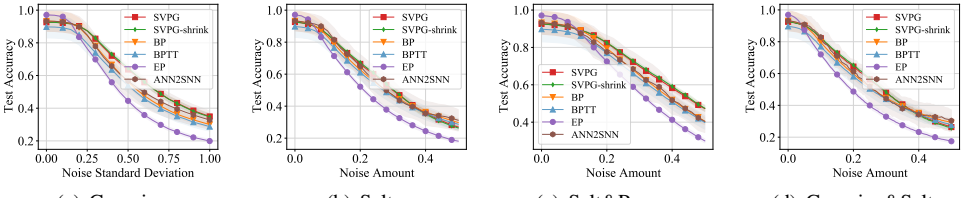
In the main text, we present the results regarding input salt noise and network Gaussian noise. Here we present the results with other noises, as shown in Figure 8. These results further support the analysis in the main text.



(a) Input Gaussian (b) Input Gaussian&Salt (c) Input Salt&Pepper (d) Network Uniform

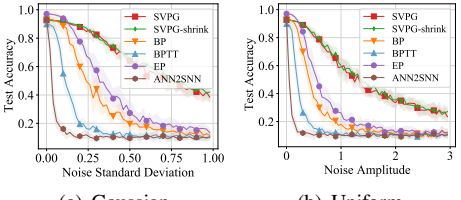
Figure 8: Additional comparison of three implementations of SVPG.

D.3 MNIST Additional Results



(a) Gaussian (b) Salt (c) Salt&Pepper (d) Gaussian&Salt

Figure 9: MNIST – Input noises.

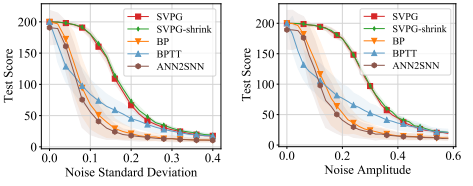


(a) Gaussian (b) Uniform

Figure 10: MNIST – Network noises.

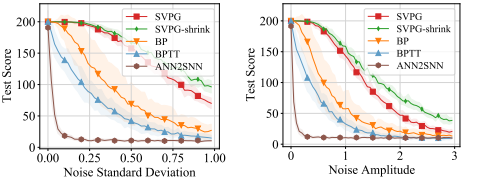
D.4 GymIP Additional Results

Here are the complete results of all the variations tested in the GymIP task. Note that the "Union" variation means the length and thickness of the pendulum change together with a fixed ratio (length:thickness=16:1).



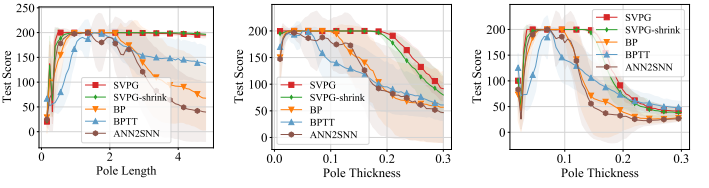
(a) Gaussian (b) Uniform

Figure 11: GymIP – Input noises.



(a) Gaussian (b) Uniform

Figure 12: GymIP – Network noises.



(a) Length (b) Thickness (c) Union

Figure 13: GymIP – Pendulum variations.

D.5 Computation Costs

The computation costs of inference and optimization of different methods in the MNIST task are shown in Table 2. The values presented are the averaged times of 10 steps. The results are gained using one NVIDIA RTX 3080 GPU. 1) For the inference period, SVPG consumes much more time than BP and EP. This is because SVPG needs to simulate a spike train during each inference step. The rate coding version of SVPG alleviates this problem and achieves a computational efficiency close to that of BP and EP. 2) For the optimization period, SVPG is the most efficient. This is because SVPG updates the parameters using only local learning rules, while other methods need backpropagation (BP and BPTT) or iterations (EP).

Time (ms)	SVPG-rate	SVPG-spike	BP	BPTT	EP
Inference	2.1	373.1	0.2	52.5	4.3
Optimization	0.2	0.5	1.6	60.1	61.1

Table 2: Computation costs.