GRSN: Gated Recurrent Spiking Neurons for POMDPs and MARL

Lang Qin[1,2], **Ziming Wang**[1,2], **Runhao Jiang**[1,2], **Rui Yan**[3*], **Huajin Tang**[1,2,4]

[1]College of Computer Science and Technology, Zhejiang University
[2]The State Key Lab of Brain-Machine Intelligence, Zhejiang University
[3]College of Computer Science and Technology, Zhejiang University of Technology
[4]MOE Frontier Science Center for Brain Science and Brain-Machine Integration, Zhejiang University
qinl@zju.edu.cn, zi_ming_wang@outlook.com, RhJiang@zju.edu.cn, Ryan@zjut.edu.cn, htang@zju.edu.cn

Abstract

Spiking neural networks (SNNs) are widely applied in various fields due to their energy-efficient and fast-inference capabilities. Applying SNNs to reinforcement learning (RL) can significantly reduce the computational resource requirements for agents and improve the algorithm's performance under resource-constrained conditions. However, in current spiking reinforcement learning (SRL) algorithms, the simulation results of multiple time steps can only correspond to a single-step decision in RL. This is quite different from the real temporal dynamics in the brain and also fails to fully exploit the capacity of SNNs to process temporal data. In order to address this temporal mismatch issue and further take advantage of the inherent temporal dynamics of spiking neurons, we propose a novel temporal alignment paradigm (TAP) that leverages the single-step update of spiking neurons to accumulate historical state information in RL and introduces gated units to enhance the memory capacity of spiking neurons. Experimental results show that our method can solve partially observable Markov decision processes (POMDPs) and multi-agent cooperation problems with similar performance as recurrent neural networks (RNNs) but with about 50% power consumption.

Code — https://github.com/StillWolf/GRSN-SNN
Extended version — https://arxiv.org/abs/2404.15597

1 Introduction

Spiking neural networks (SNNs) possess unique spatiotemporal dynamics and all-or-none firing characteristics, enabling them to perform low-power and high-speed inference on neuromorphic hardware (Roy, Jaiswal, and Panda 2019; Pei et al. 2019). With the continuous improvement of SNN training algorithms, the learning capability and network scalability of SNNs are gradually approaching those of artificial neural networks (ANNs) (Wang et al. 2023; Lian et al. 2023; Qin et al. 2023). Therefore, in recent years, there have been more and more efforts to explore SNNs as alternatives to ANNs in the field of deep reinforcement learning (DRL), aiming to reduce the computational cost and inference latency of intelligent agents.

*Author for correspondence.

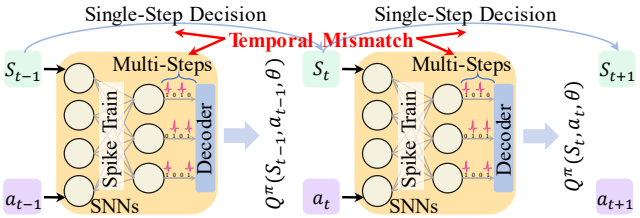

Figure 1: Temporal mismatch issue in SRL algorithms. The figure shows the basic model of SRL methods; it uses spike results of multiple time steps to decode for one-step value function calculation or action selection in RL.

In SNN learning algorithms, firing rates are often used to calculate continuous values due to the discrete nature of spikes. However, the firing rates, which are in $[0, 1]$, are difficult to map onto continuous values (such as continuous actions and Q-values) in RL that do not have value range limitations. There are generally three types of approaches to address or avoid this issue in order to achieve deep spiking reinforcement learning (SRL): (1) ANN-SNN conversion methods for RL; (2) a hybrid framework for the actor-critic (AC) method; and (3) directly trained SNNs for RL.

The conversion algorithms (Patel et al. 2019; Tan, Patel, and Kozma 2021) avoid the aforementioned mapping problem by using a pre-trained ANN-RL model to transform the SNN model, thus avoiding direct involvement in the learning process of RL algorithms. However, the conversion algorithms often require high simulation time steps due to the computational precision requirement, resulting in increased energy consumption and inference delay. To reduce simulation steps, a hybrid framework (Tang, Kumar, and Michmizos 2020; Tang et al. 2020; Zhang et al. 2022) for the AC method has been proposed. This approach combines spiking actor networks and artificial critic networks and utilizes a gradient descent algorithm for collaborative training. The critic networks that require precise calculations are completed by ANN, while SNN is only used for selecting actions. This hybrid framework successfully reduces the need for simulation time steps, thereby reducing energy consumption and inference delay.

Clearly, the hybrid framework method is only applicable to RL algorithms based on the AC method. Additionally,

similar to conversion algorithms, the hybrid framework method still relies on ANN to accomplish RL tasks. To enhance the generalizability of the algorithms and eliminate reliance on ANN, directly trained SNNs for RL (Chen et al. 2022; Liu et al. 2022; Sun, Zeng, and Li 2022; Qin, Yan, and Tang 2023) have been proposed. These algorithms directly train SNNs through surrogate gradient learning (Neftci, Mostafa, and Zenke 2019), and use suitable coders to handle computations involving continuous values. These methods do not rely on ANNs and can complete tasks in a few time steps. However, such methods rely more on the design and selection of coders and also have a trade-off between performance and latency.

Reinforcement learning algorithms are fundamentally designed to address sequential decision problems; they can synergize effectively with SNNs that inherently capture temporal sequence dynamics. However, the aforementioned existing SRL methods still use multiple time steps to decode the spike information to obtain continuous values, which leads to a serious temporal mismatch problem (Fig. 1). This temporal mismatch between multiple time steps and single-step decisions not only deviates from the real dynamics in the brain but also impairs the ability of SNNs to process temporal data.

To address this issue, we propose a novel SRL paradigm that aligns the single-step updating of spiking neurons with the single-step decisions in RL, enabling the sequence decision problem to be solved within a whole simulated time window. Considering the weak temporal correlations between time steps of the original spiking neurons and the inability to guarantee accuracy after temporal alignment, we further introduced gated units to enhance the long- and short-term memory capabilities of neurons, ensuring the effectiveness of the proposed paradigm. In order to test the performance of the proposed gated recurrent spiking neurons (GRSN) and temporal alignment paradigm (TAP), we conducted experiments in partially observable (PO) and multi-agent environments. Our main contributions are summarized as follows:

- We pointed out the issue of temporal mismatch in current SRL algorithms and proposed a novel temporal alignment paradigm (TAP) for SRL to solve this issue.
- We designed a novel gated recurrent spiking neuron (GRSN) with enhanced long- and short-term memory capabilities, which can be applied to the proposed TAP.
- To our best knowledge, this work is the first to use SNNs to address POMDPs and multi-agent reinforcement learning (MARL) problems. We verified the effectiveness of SNNs in PO and multi-agent environments.
- Experimental results show that GRSN can outperform original spiking neurons in benchmark environments and can achieve similar performance as RNN-RL with about 50% energy consumption.

2 Related Works

2.1 Spiking Reinforcement Learning

The development of SRL can be mainly divided into three periods: (1) the basic research of SNNs and RL (Williams 1992; Bohté, Kok, and Poutré 2002); (2) the synaptic-plasticity-based SRL algorithm (Seung 2003; Urbanczik and Senn 2009; Frémaux, Sprekeler, and Gerstner 2010); and (3) the combination of deep RL and SNNs (Zhang et al. 2021; Liu et al. 2022; Qin, Yan, and Tang 2023). Early-stage works often study the combination of synaptic plasticity and RL theory, which aims to reveal how reward mechanisms in the brain correlate and combine with RL algorithms. In the later stage, with the development of DRL, the SRL algorithm focuses on applying SNNs to DRL, aiming for better performance.

2.2 Recurrent Spiking Neural Networks

Due to the inherent temporal nature of spiking neurons, there have been many attempts to use recurrent spiking neural networks for temporal data processing. Some work (Lotfi-Rezaabad and Vishwanath 2020; Rao et al. 2022) has designed spike-based long short-term memory networks and tested them on audio or text datasets. Other work (Yin, Corradi, and Bohté 2020; Ponghiran and Roy 2022; Liu et al. 2023) is to explore the long-term dependence of spiking neurons and complete structure optimization or time series forecasting tasks.

2.3 Neural Coding Frameworks

Neural encoding is also one of the important directions of SRL. Many important coding frameworks (Zhang et al. 2023; Ma et al. 2006) can realize the coding and representation of the probability distribution of the features through neural populations in the spatial dimension. These methods can effectively reduce the analog-digital conversion error in the coding process and achieve an accurate expression of the value function.

2.4 Applications of SRL

There are many downstream applications of SRL, including neuromorphic robot control (Polykretis et al. 2022), multi-agent collaboration (Saravanan et al. 2021), embodied intelligence (Bartolozzi, Indiveri, and Donati 2022), etc. These practical application directions also prove the importance of SRL algorithm research.

3 Preliminaries

3.1 Notations

For all variables, superscripts refer to the number of layers in the neural network, and subscripts refer to their time steps in the time window. For example, $\boldsymbol{u}_t^l$ denotes the membrane potential in layer l at the t-th time step. We follow the conventions representing vectors and matrix with bold italic letters and bold capital letters, respectively, such as $\boldsymbol{o}$ and $\boldsymbol{W}$. We use $\odot$ to signify element-wise multiplication for two vectors.

3.2 Leaky Integrate-and-Fire Model

Different from traditional neural networks, SNNs use binary spikes as information carriers. In order to compensate for the lack of binary spikes in information expression, the time dimension, or latency, is introduced into SNNs. SNNs

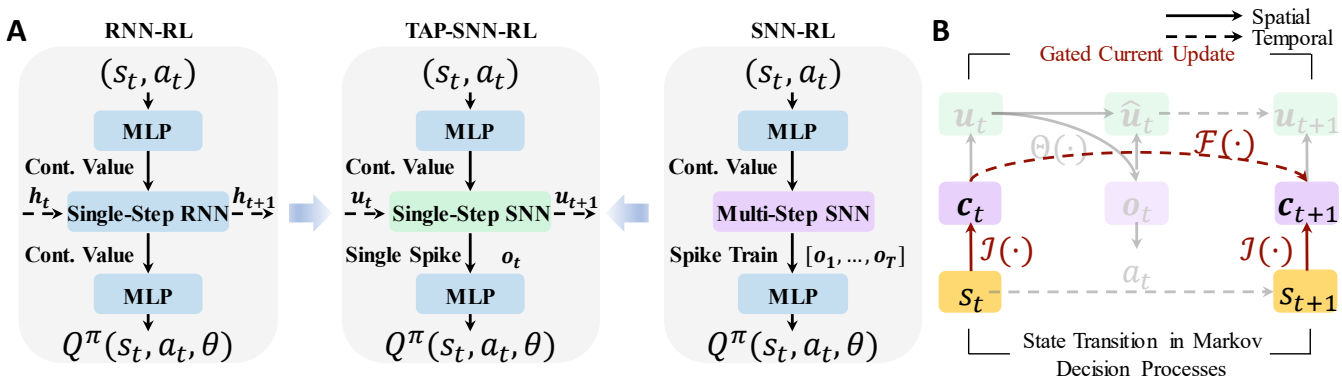

Figure 2: Overview of TAP and GRSN. **A.** Basic network model. TAP is more similar to the RNN-RL method. We only replace the single-step RNN with the single-step SNN. TAP-SNN-RL only needs to calculate single-step updates and use one spike for calculation, while SNN-RL without TAP needs to loop multiple times and use a spike train for calculation. **B.** Dynamics of the GRSN. The red part shows the recurrent connection and gated unit. $\mathcal{F}$ and $\mathcal{I}$ represent the forget gates and input gates, respectively. The background part represents the update rules which are the same as LIF neurons.

accept event streams as input, and the forward propagation of SNNs is repeated for T time steps in the temporal dimension to calculate the output spike trains. Therefore, the basic computing unit of SNNs, spiking neurons, has unique spatio-temporal dynamics. Here we introduce the widely used Leaky Integrate-and-Fire (LIF) neuron (Wu et al. 2018) as the benchmark in this work. In each time step t, the membrane potential $\boldsymbol{u}_t^l$ of spiking neurons at the l-th layer will integrate the input current $\boldsymbol{c}_t^l$ and the decay voltage $\beta\hat{\boldsymbol{u}}_{t-1}^l$:

$$\boldsymbol{u}_t^l = \beta\hat{\boldsymbol{u}}_{t-1}^l + (1-\beta)\boldsymbol{c}_t^l \tag{1}$$

$$\boldsymbol{c}_t^l = \boldsymbol{W}^l\boldsymbol{o}_t^{l-1} + \boldsymbol{b}^l \tag{2}$$

Membrane potential will be reset to u_r (default set to 0) when a spike is emitted at the last time step:

$$\hat{\boldsymbol{u}}_t^l = u_r\boldsymbol{o}_t^l + (1-\boldsymbol{o}_t^l)\boldsymbol{u}_t^l \tag{3}$$

The neurons will emit output spikes $\boldsymbol{o}_t^l$ whenever the membrane potential $\boldsymbol{u}_t^l$ exceeds the threshold ϑ:

$$\boldsymbol{o}_t^l = \Theta(\boldsymbol{u}_t^l - \vartheta) \tag{4}$$

Where $\Theta(x)$ is the Heaviside function:

$$\Theta(x) = \begin{cases} 1, if \quad x \geq 0 \\ 0, otherwise \end{cases} \tag{5}$$

3.3 Surrogate Gradient

Due to the non-differentiability of the binary firing function $\Theta(x)$, SNNs often employ surrogate gradients (Neftci, Mostafa, and Zenke 2019) during backpropagation (BP). We adopt the arc-tangent function to replace the derivative:

$$\begin{aligned} h(x) &= \frac{1}{\pi}arctan(\frac{\pi}{2}\alpha x) + \frac{1}{2} \\ \Theta'(x) \triangleq h'(x) &= \frac{\alpha}{2[1+(\frac{\pi}{2}\alpha x)^2]} \end{aligned} \tag{6}$$

where α is a hyperparameter that controls the width of the surrogate function. With the surrogate gradients for firing functions, we can directly use the BP method to train SNNs.

4 Method

4.1 Temporal Alignment Paradigm

Reinforcement learning tasks can be modeled as Markov decision processes (MDP) $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, scalar reward function $\mathcal{R}$, transition dynamics p, and discount factor γ (Sutton and Barto 2018). RL agents are controlled by policy π, with the goal of maximizing the expected discounted return $\mathbb{E}_\pi[\sum_{t=0}^{\infty}\gamma^t r_{t+1}]$. According to the Bellman equation (Bellman 1966), we can obtain the iteration form of the action value function:

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi[R_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1})] \tag{7}$$

Considering we use the single-step TD-error in RL, the updating of the action value function can be described as:

$$\begin{aligned} Q(s_t, a_t) &= \frac{1}{\gamma}Q(s_{t-1}, a_{t-1}) + \frac{-1}{\gamma}R_t \\ &= \frac{1}{\gamma}Q(s_{t-1}, a_{t-1}) + (1-\frac{1}{\gamma})\frac{R_t}{1-\gamma} \end{aligned} \tag{8}$$

Observing Eq. 8 and Eq. 1, we can find that the state transitions in MDPs exhibit a striking resemblance to the state changes of spiking neurons. Therefore, the key to solving the temporal mismatch problem is to utilize this similarity. We propose a novel SRL paradigm according to this similarity: within the framework of RL algorithms, the application of SNNs in a manner similar to RNNs (Fig. 2A), aligning each step of spiking neuron updates $\boldsymbol{u}_t \rightarrow \boldsymbol{u}_{t+1}$ with each state transition $s_t \rightarrow s_{t+1}$ in an MDP. The advantage of this approach lies in the natural utilization of the temporal structure of SNNs while significantly reducing the number of time steps required for the entire task. Fig. 2 shows the proposed TAP and the detailed dynamics of spiking neurons.

4.2 Gradient Analysis of Spiking Neurons

In Section 3.2, we introduce the basic computational unit of SNNs: LIF neurons. It uses a constant leaky factor β to

play the roles of forget gate and input gate. The use of this constant factor results in weaker temporal correlations in the neuron's internal state ($\boldsymbol{u}_t$), making it challenging to handle data with long-term dependencies. In order to enhance the memory capacity of SNNs in the temporal domain, we need to analyze and optimize the LIF neuron.

According to Eq. 1-5, it can be observed that the LIF neuron has an internal state and an external input: membrane potential $\boldsymbol{u}_t$ and input current $\boldsymbol{c}_t$. To compare the importance of these two parameters in backpropagation, we need to calculate the gradient of the loss function to connection weight. Suppose $\mathcal{L}$ is the loss function that we would like to minimize. According to Eq. 1-5 and the chain rule, its gradient is:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^l} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \boldsymbol{o}_t^l} \frac{\partial \boldsymbol{o}_t^l}{\partial \boldsymbol{u}_t^l} \frac{\partial \boldsymbol{u}_t^l}{\partial \boldsymbol{W}^l} \tag{9}$$

T is the total number of time steps. The first term $\frac{\partial \mathcal{L}}{\partial \boldsymbol{o}_t^l}$ in Eq. 9 is determined by the decoder and loss function and does not affect the flow of the gradient. According to Eq. 4 and Eq. 6, the second term could be derived as:

$$\frac{\partial \boldsymbol{o}_t^l}{\partial \boldsymbol{u}_t^l} = \frac{\partial \Theta(\boldsymbol{u}_t^l - \vartheta)}{\partial \boldsymbol{u}_t^l} = h'(\boldsymbol{u}_t^l - \vartheta) \tag{10}$$

For brevity, let us go straight to the final gradient calculation (please refer to the appendix for the detailed derivation process):

$$\begin{aligned} \frac{\partial \boldsymbol{u}_t^l}{\partial \boldsymbol{W}^l} =& (1-\beta) \sum_{i=1}^{t} \beta^i \prod_{j=1}^{i} \delta_{t-j} \frac{\partial \boldsymbol{c}_{t-i}^l}{\partial \boldsymbol{W}^l} + \\ & \prod_{k=1}^{t} \beta \delta_{t-k} + (1-\beta) \frac{\partial \boldsymbol{c}_t^l}{\partial \boldsymbol{W}^l} \end{aligned} \tag{11}$$

When the variable subscript $t = 0$, its value is the artificially set initial value.

To analyze the contributions of parameters ($\boldsymbol{u}_t$ and $\boldsymbol{c}_t$) of spiking neurons to the gradient, we analyzed Eq. 11 under commonly used parameter settings ($\beta = 0.5, \vartheta = 1, \alpha = 2, u_r = 0$): The first term in Eq. 9 depends on the decoding method, and the second term in Eq. 9 depends on the SG function. Both of them do not affect the direction of gradient flow. Therefore, the third term ($\frac{\partial \boldsymbol{u}_t^l}{\partial \boldsymbol{W}^l}$) is the most significant. By analyzing the monotonicity (presented in the appendix) of Eq. 11, we can observe that as T increases, the term $\prod_{i=1}^{T} \beta \delta_t$ tends to converge to zero. Hence, the last term $(1-\beta)\frac{\partial \boldsymbol{c}_t^l}{\partial \boldsymbol{W}^l}$ in Eq. 11 affects the gradient mostly. Based on the above analysis, the contribution of current $\boldsymbol{c}$ to the gradient is greater than that of membrane potential $\boldsymbol{u}$.

To further verify this conclusion, we conducted experiments in *CartPole-V* (see Section 5) environment and visualized the parameter distribution during the training process. According to the results (Fig. 3), the maximum absolute value of the term $\beta\delta_t$ is around 0.5 and less than 1, and most of the membrane potential values are distributed around 0. Hence, the second term in Eq. 11 will tend to 0.

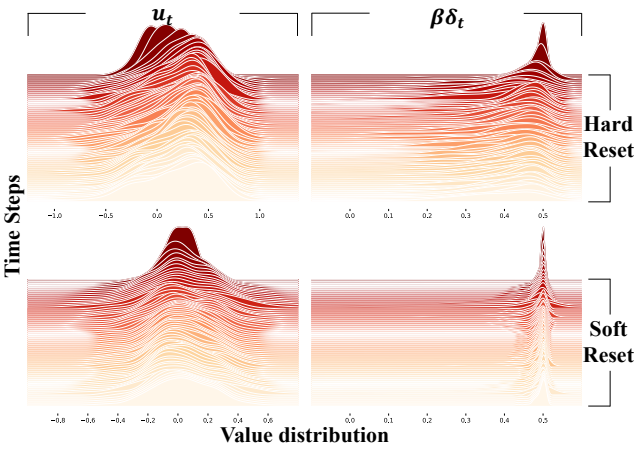

Figure 3: Distributions of membrane potential $\boldsymbol{u}_t$ and $\beta\delta_t$. The y-axis represents the time step, and the x-axis represents the value of the parameter. The probability density curve corresponding to each time step represents the parameter distribution of all neurons in the single-layer network.

Based on the analysis, the gradient of the loss function $\mathcal{L}$ is mainly transmitted through the input current $\boldsymbol{c}$. Therefore, adding gated recurrent connections to the input current $\boldsymbol{c}$ can enhance the memory capability of spiking neurons more effectively than membrane potential $\boldsymbol{u}$.

4.3 Gated Recurrent Spiking Neurons

Optimizing the dynamics of neurons and improving memory ability is mainly divided into three steps. The first step is to change the decay factor β to a learnable parameter, which can be jointly optimized with the entire network. The learnable decay factor can dynamically adjust the forgetting ability of neurons during the training process and adaptively find the optimal value (Rathi and Roy 2023). Then, we use a soft reset instead of a hard reset (the gradient of the loss function is still mainly transmitted through current $\boldsymbol{c}$ in the soft reset setting; see Fig. 3 and appendix for details). Compared to the hard reset method, soft reset can reduce temporal information loss, and it has better performance when the network is shallow (Ledinauskas et al. 2020). Finally, we added recurrent connections and gating functions to enhance the temporal correlation of the neuron's internal states $\boldsymbol{c}$.

We named the proposed neuron model gated recurrent spiking neurons (GRSN). Its update rules, firing rules, and definitions of Heaviside function are consistent with those of LIF neurons (Eq. 1, Eq. 4 and Eq. 5). We use soft reset to replace the hard reset (Eq. 3):

$$\hat{\boldsymbol{u}}_t^l = \boldsymbol{u}_t^l - \vartheta \boldsymbol{o}_t^l \tag{12}$$

GRSN also adds recurrent connections at the current:

$$\boldsymbol{c}_t^l = \mathcal{F}(\boldsymbol{o}_t^{l-1}) \odot \boldsymbol{c}_{t-1}^l + [1 - \mathcal{F}(\boldsymbol{o}_t^{l-1})] \odot \mathcal{I}(\boldsymbol{o}_t^{l-1}) \tag{13}$$

where $\mathcal{F}(\cdot)$ is the forgetting gate that regulates the proportion of forget and input, and $\mathcal{I}(\cdot)$ is the input gate that processes

Alg.-Net. \ Env.	Learning Paradigm	CartPole		Pendulum	
		-V	-P	-V	-P
TD3-MLP *	Actor-Critic	21.6 ± 1.49	21.2 ± 1.95	-1502.8 ± 61.09	-1380.1 ± 47.45
TD3-GRU *	Recurrent RL †	**200.0 ± 0.00**	196.7 ± 6.30	**-181.3 ± 15.47**	-203.5 ± 16.36
TD3-LIF	**TAP**	**200.0 ± 0.00**	152.2 ± 21.21	-942.4 ± 92.19	-824.8 ± 178.80
TD3-GRSN	**TAP**	**200.0 ± 0.00**	**199.9 ± 0.20**	-189.4 ± 17.64	**-195.8 ± 43.99**

* Reproduction results of (Ni, Eysenbach, and Salakhutdinov 2022).
† Model-Free RL model (Ni, Eysenbach, and Salakhutdinov 2022) that contains embedding layers of states/actions/rewards.

Table 1: Performance Comparison on PO Environment

Algorithm 1: GRSN with temporal alignment paradigm

Input: States sequence $< s_1, s_2, ..., s_T >$
Parameter: Total steps T, total layers L
Output: Values sequence $< q_1, q_2, ..., q_T >$
initialize the hidden states $\boldsymbol{u}_0^1 = \hat{\boldsymbol{u}}_0^1 = \mathbf{0}, \boldsymbol{c}_0^1 = \mathbf{0}$
for $t = 1$ to T **do**
  initialize the input spike $\boldsymbol{o}_t^0 = encoder(s_t)$
  for $l = 1$ to L **do**
    update $\boldsymbol{c}_t^l$ and $\boldsymbol{u}_t^l$ // Eq 13, Eq 1
    firing and computing output $\boldsymbol{o}_t^l$ // Eq 4
    reset the potential $\hat{\boldsymbol{u}}_t^l$ // Eq 12
  end for
  compute the values $q_t = decoder(\boldsymbol{o}_t^L)$
end for
return $< q_1, q_2, ..., q_T >$

Notes: The encoder and decoder are implemented by MLP, similar to the embedding layers in RL.

input signals. Both $\mathcal{F}(\cdot)$ and $\mathcal{I}(\cdot)$ are composed of a fully connected (FC) layer and an activation function. Their detailed definitions are shown as follows:

$$\begin{aligned} \mathcal{F}(\boldsymbol{x}) &= \sigma(\boldsymbol{W}_f \boldsymbol{x} + \boldsymbol{b}_f) \\ \mathcal{I}(\boldsymbol{x}) &= ReLU(\boldsymbol{W}_i \boldsymbol{x} + \boldsymbol{b}_i) \end{aligned} \tag{14}$$

where σ is the sigmoid function.

Fig. 2B shows the dynamics of GRSN in the temporal alignment paradigm, red lines indicate the recurrent connections. The pseudocode of GRSN with the temporal alignment paradigm is shown in Algorithm 1.

5 Experiments

In the TAP, SNN behaves more like an RNN rather than a traditional DNN. Spiking neurons calculate the value function by accumulating information from multiple previous consecutive states. This configuration is often employed in RL to address POMDPs. Consequently, to validate the effectiveness and versatility of the proposed GRSN, we conducted experiments in both PO environments and multi-agent cooperative environments. The detailed parameters and settings of the experiment can be found in the appendix.

5.1 Partially Observable Classic Control Tasks

Environments & Experimental Settings The *Pendulum* and *CartPole* tasks are classic control tasks for evaluating

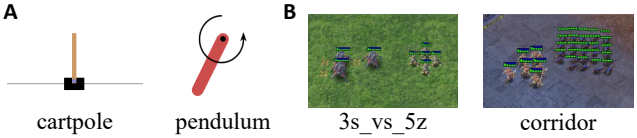

Figure 4: **A.** Diagram of the pendulum and cartpole problem. The agent needs to apply appropriate forces to control the system to reach the target state, in order to get rewards. **B.** Game scenarios of two different SMAC maps. Agents need to defeat enemies to achieve high rewards and win the game.

RL algorithms. The cart-pole problem described in (Barto, Sutton, and Anderson 1983) aims to balance the pole by applying forces in the left and right directions on the cart (Figure 4A). The inverted pendulum swingup problem is based on the classic problem in control theory. Its goal is to apply torque on the free end to swing it into an upright position, with its center of gravity right above the fixed point (Figure 4A). We conducted experiments in PO cases of these two control tasks (Han, Doya, and Tani 2020), in which only velocities or positions could be observed. We use suffix names (-P/-V) to distinguish these two settings.

We follow the experimental settings proposed in (Ni, Eysenbach, and Salakhutdinov 2022) and use SAC (Haarnoja et al. 2018) for discrete environments and TD3 (Fujimoto, van Hoof, and Meger 2018) for continuous environments. For networks, we use single-layer gated recurrent units (GRU) (Chung et al. 2014) for RNNs, two fully connected layers for MLP, and single-layer LIF/GRSN for SNNs. We conducted independent experiments on five different random seeds for each method and tested each final model ten times to eliminate the interference of randomness. Experiment details can be found in the appendix.

Results To demonstrate the superiority of the TAP and GRSN in multiple aspects, we selected three different networks and paradigm settings as baselines (Table 1). From the results, TAP effectively utilizes temporal information, and its performance far exceeds that of traditional algorithms using MLP. On the other hand, GRSN enhances the temporal correlation of spiking neurons, thereby further improving performance. The combination of TAP and GRSN equals or even surpasses the state-of-the-art RNN-RL method. LIF and MLP, which are baselines, are weaker than GRSN and RNNs in terms of temporal correlation, so their performance

Scenarios	Difficulty	QMIX-GRU †		QMIX-GRSN	
		Mean reward	Win rate (%)	Mean reward	Win rate (%)
8m	Easy	19.8 ± 0.26	97.6 ± 3.49	$\mathbf{20.0 \pm 0.12}$	$\mathbf{99.4 \pm 1.48}$
2s3z	Easy	$\mathbf{19.9 \pm 0.18}$	$\mathbf{97.9 \pm 2.84}$	$\mathbf{19.9 \pm 0.20}$	97.0 ± 3.57
8m_vs_9m	Hard	19.4 ± 0.69	91.6 ± 8.73	$\mathbf{19.6 \pm 0.35}$	$\mathbf{94.6 \pm 4.54}$
3s_vs_5z	Hard	21.0 ± 0.44	$\mathbf{97.1 \pm 3.46}$	$\mathbf{21.4 \pm 0.60}$	93.0 ± 5.48
27m_vs_30m	Super Hard	19.2 ± 0.51	79.4 ± 12.19	$\mathbf{19.9 \pm 0.14}$	$\mathbf{96.4 \pm 3.74}$
MMM2	Super Hard	17.6 ± 3.76	75.2 ± 37.93	$\mathbf{18.5 \pm 0.73}$	$\mathbf{83.6 \pm 8.17}$

† Reproduction results of (Hu et al. 2021).

Table 2: Experimental results in SMAC environment

Enviroment	*Pendulum-P*				*SMAC-8m*			
Paradigm	w/ TAP		w/o TAP †		w/ TAP		w/o TAP †	
	LIF	GRSN	LIF	GRSN	LIF	GRSN	LIF	GRSN
Mean Reward ↑	-824.8	**-195.8**	-1069.9	-235.4	15.3	**19.9**	16.9	19.7
Standard Deviation ↓	178.80	**43.99**	65.85	44.16	1.59	**0.02**	1.38	0.31
Test Win Rate ↑ (%)	-	-	-	-	50.4	**99.4**	64.6	96.5
Total Time Steps* ↓	**64**	**64**	256	256	**26**	**26**	104	104
Training Time ↓ (h)	3.55	**3.42**	7.26	10.18	12.91	**12.76**	19.52	18.57

† For all experiments w/o TAP, SNNs use repeated input and rate coding, and the time step is set to $T = 4$.
* In the SMAC environment, the average episode length is used instead of time steps.

Table 3: Ablation Study

and convergence rate are far inferior to the other two networks.

5.2 StarCraft Multi-Agent Challenge

Environments & Experimental Settings StarCraft Multi Agent Challenge (SMAC) is a benchmark environment for evaluating MARL algorithms (Samvelyan et al. 2019). SMAC is based on the popular real-time strategy (RTS) game StarCraft II, and it contains multiple micro StarCraft II scenarios (Figure 4B). In SMAC, the overall goal is to maximize the win rate for each battle scenario. Each agent in SMAC has a circular field of view centered on itself. With such a limited field of view, each agent is faced with a partially observable environment. The action space is discrete and includes *move[direction]*, *attack[enemy_id]*, *stop*, and *no-op*. Healing units use *heal[agent_id]* actions instead of *attack[enemy_id]*. The SMAC reward includes several parts: hit-point damage dealt and enemy units killed, together with a special bonus for winning the battle.

The QMIX (Rashid et al. 2018) is a classic value-based multi-agent collaboration algorithm. QMIX adopts the centralized training distributed execution (CTDE) mode, and each agent uses RNNs as main networks. Therefore, the proposed GRSN and TAP can be well adapted to the QMIX algorithm. We follow the experimental and parameter settings of QMIX and select SMAC maps of different difficulty levels for the experiment. Detailed parameters and settings are also listed in the appendix.

Results To evaluate the performance more comprehensively, we selected several SMAC maps with different difficulty levels for experiments. We conducted 5 independent experiments in each environment, training 10 million steps per experiment. We use the GRU-based QMIX algorithm as the baseline, and the experimental results are shown in Table 2. From the experimental results, it can be seen that the GRSN-based QMIX algorithm surpasses the original GRU-based QMIX algorithm in most environments, which proves that GRSN achieves or even exceeds GRU in the processing of temporal information. In addition, the advantages of GRSN are more obvious in difficult scenarios. This also proves that GRSNs using spikes have better adaptability to the environment than traditional RNNs.

5.3 Ablation Study

In Section 4, we propose the TAP that can solve the temporal mismatch problem and greatly reduce time steps, and the GRSN that can enhance the temporal association of spiking neurons and improve performance. To demonstrate the impact of these two on the training speed and algorithm performance more intuitively, we conducted multiple groups of ablation experiments. We ran experiments in both classical control tasks and SMAC environments. Similar to the previous setting, we conducted five independent experiments at each of the different settings and averaged them as the final results.

Temporal Alignment Paradigm The TAP solves the mismatch problem, thus significantly reducing the number of

Enviroment		GRU			GRSN			Mixer
		MLP (K)	RNN (K)	Energy (KpJ)	MLP (K)	SNN (K)	Energy (KpJ)	
PO-Ctrl.	*Pendulum-P*	116.39	136.70	1164.22	108.19	9.73	**506.44**	-
	Pendulum-V	116.24		1163.56	108.05	11.16	**507.07**	
	CartPole-P	116.74		1165.85	108.55	6.98	**505.91**	
	CartPole-V	116.74		1165.85	108.55	7.33	**505.60**	
MARL	*8m*	7.57	24.96	149.62	14.98	0.01	**68.90**	51.20
	2s3z	6.99		146.97	22.85	0.01	**105.11**	35.75
	8m_vs_9m	8.02		151.69	15.36	0.06	**70.71**	53.31
	3s_vs_5z	4.81		136.95	12.42	0.03	**57.15**	21.60
	27m_vs_30m	24.74		228.62	30.72	0.07	**141.38**	283.11
	MMM2	14.35		180.84	29.76	0.01	**136.90**	84.93

Table 4: Energy Consumption Estimation

time steps and training and inference time. From Table 3, the algorithm using TAP reduces the time step by T times, where $T = 4$ is the time window size of the baseline spiking neuron. In addition, TAP has also shortened the training time by about half while maintaining performance, greatly improving the efficiency of algorithms.

GRSN vs. LIF In Table 3, we can also compare the GRSN and LIF neurons. GRSN has better performance in different environmental and paradigm settings. However, due to the additional gating function, the training time of GRSN will be slightly longer than that of LIF. In addition, GRSN also demonstrated the most stable performance, with the lowest standard deviation in several independent experiments.

5.4 Energy Consumption Estimation

SNNs use discrete spikes for information transmission, so their energy consumption is estimated differently than that of ANNs. We follow the convention of the neuromorphic computing community (Wu et al. 2023; Wang et al. 2023) by counting the total synaptic operations (SOP) to estimate the energy consumption of SNN models. Specifically, the SOP for SNNs correlates with the neurons' firing rate, fan-out f_{out}, and time window size T:

$$SOP = \sum_{t=1}^{T} \sum_{l=1}^{L-1} \sum_{j=1}^{N^l} f_{out}^l[j] \boldsymbol{o}_t^l[j] \tag{15}$$

where L is the total number of layers, N^l is the number of neurons in layer l, f_{out} is the number of outgoing connections, and $\boldsymbol{o}_t^l[j]$ denotes the output spike of the j-th neuron in layer l at the t-th time step.

As for ANNs, SOP is only related to network structure. It is defined as:

$$SOP = \sum_{l=1}^{L} f_{in}^l N^l \tag{16}$$

where f_{in}^l represents the number of incoming connections to each neuron in layer l, L and N^l are consistent with their definitions in SNNs.

SNNs use efficient addition operations, while ANNs use more expensive multiply-accumulate operations. According to (Han et al. 2015), we measure 32-bit floating-point addition operations by $0.9pJ$ per operation and 32-bit floating-point multiply-accumulate operations by $4.6pJ$ per operation. Here, we randomly select 1024 samples to estimate the average SOP and energy consumption for SNNs.

Due to the additional computational overhead introduced by recurrent connections, we use the same power consumption standard ($4.6pJ$ per operation) as the ANN for the non-spiked recurrent connections inside spiking neurons. Table 4 shows the results in different environments. In particular, the MLP column counts the synaptic operations of all embedding layers and recurrent connections in GRSNs. It can be seen that SNNs can reduce energy consumption by up to about 3800 times compared to RNNs. The total energy consumption is also reduced by about 50%, which effectively reduces the agent's resource requirements.

6 Conclusion

In this paper, we introduce the temporal mismatch problem in SRL algorithms and propose the temporal alignment paradigm (TAP) and gated recurrent spiking neurons (GRSN) to solve this problem. Specifically, TAP allows the single-step update of spiking neurons to correspond to the single-step decision-making of MDP to solve the mismatch problem and significantly reduces time steps. On the other hand, GRSN enhances the temporal correlation of spiking neurons, thereby improving their ability to capture historical information and compensating for the performance degradation caused by time step reduction. Extensive experiments show that TAP can reduce the time steps by several times and half the training time, which effectively solves the temporal mismatch problem. At the same time, GRSN can also improve the performance of spiking neurons, making SNNs equal to or even exceed the traditional RNNs. Energy estimation proves that our method can reduce the energy consumption by about 50%, which provides a new algorithm option for the control task in the energy-constrained scenario.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grant Nos. 62236007, 62276235, and 32441113) and the National Key Research and Development Program of China (Grant No. 2024YDLN0005).

The authors would also like to acknowledge anonymous reviewers and chairs for providing insightful comments to help improve this work.

References

Barto, A. G.; Sutton, R. S.; and Anderson, C. W. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5): 834–846.

Bartolozzi, C.; Indiveri, G.; and Donati, E. 2022. Embodied neuromorphic intelligence. *Nature Communications*, 13: 1024.

Bellman, R. 1966. Dynamic programming. *Science*, 153(3731): 34–37.

Bohté, S. M.; Kok, J. N.; and Poutré, J. A. L. 2002. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48: 17–37.

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

Chung, J.; Gülçehre, Ç.; Cho, K.; and Bengio, Y. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR*, abs/1412.3555.

Frémaux, N.; Sprekeler, H.; and Gerstner, W. 2010. Functional requirements for reward-modulated spike-timing-dependent plasticity. *Journal of Neuroscience*, 30: 13326–13337.

Fujimoto, S.; van Hoof, H.; and Meger, D. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning*, 1582–1591.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning*, 1856–1865.

Han, D.; Doya, K.; and Tani, J. 2020. Variational Recurrent Models for Solving Partially Observable Control Tasks. In *Proceedings of the 8th International Conference on Learning Representations*.

Han, S.; Pool, J.; Tran, J.; and Dally, W. J. 2015. Learning both Weights and Connections for Efficient Neural Networks. *CoRR*, abs/1506.02626.

Hu, J.; Jiang, S.; Harding, S. A.; Wu, H.; and wei Liao, S. 2021. Rethinking the Implementation Tricks and Monotonicity Constraint in Cooperative Multi-Agent Reinforcement Learning. *CoRR*, abs/2102.03479.

Ledinauskas, E.; Ruseckas, J.; Jursenas, A.; and Burachas, G. 2020. Training Deep Spiking Neural Networks. *CoRR*, abs/2006.04436.

Lian, S.; Shen, J.; Liu, Q.; Wang, Z.; Yan, R.; and Tang, H. 2023. Learnable Surrogate Gradient for Direct Training Spiking Neural Networks. In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence*, 3002–3010.

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

Liu, Q.; Long, L.; Peng, H.; Wang, J.; Yang, Q.; Song, X.; Riscos-Núñez, A.; and Pérez-Jiménez, M. J. 2023. Gated Spiking Neural P Systems for Time Series Forecasting. *IEEE Transactions on Neural Networks and Learning Systems*, 34: 6227–6236.

Lotfi-Rezaabad, A.; and Vishwanath, S. 2020. Long Short-Term Memory Spiking Networks and Their Applications. In *Proceedings of the International Conference on Neuromorphic Systems*, 3:1–3:9.

Ma, W. J.; Beck, J. M.; Latham, P. E.; and Pouget, A. 2006. Bayesian inference with probabilistic population codes. *Nature Neuroscience*, 9: 1432–1438.

Neftci, E. O.; Mostafa, H.; and Zenke, F. 2019. Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36: 51–63.

Ni, T.; Eysenbach, B.; and Salakhutdinov, R. 2022. Recurrent Model-Free RL Can Be a Strong Baseline for Many POMDPs. In *Proceedings of the 39th International Conference on Machine Learning*, 16691–16723.

Patel, D.; Hazan, H.; Saunders, D. J.; Siegelmann, H. T.; and Kozma, R. 2019. Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to Atari Breakout game. *Neural Networks*, 120: 108–115.

Pei, J.; Deng, L.; Song, S.; Zhao, M.; Zhang, Y.; Wu, S.; Wang, G.; Zou, Z.; Wu, Z.; He, W.; Chen, F.; Deng, N.; Wu, S.; Wang, Y.; Wu, Y.; Yang, Z.; Ma, C.; Li, G.; Han, W.; Li, H.; Wu, H.; Zhao, R.; Xie, Y.; and Shi, L. 2019. Towards artificial general intelligence with hybrid Tianjic chip architecture. *Nature*, 572: 106–111.

Polykretis, I.; Tang, G.; Balachandar, P.; and Michmizos, K. P. 2022. A Spiking Neural Network Mimics the Oculomotor System to Control a Biomimetic Robotic Head Without Learning on a Neuromorphic Hardware. *IEEE Transactions on Medical Robotics and Bionics*, 4: 520–529.

Ponghiran, W.; and Roy, K. 2022. Spiking Neural Networks with Improved Inherent Recurrence Dynamics for Sequential Learning. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, 8001–8008.

Qin, L.; Wang, Z.; Yan, R.; and Tang, H. 2023. Attention-Based Deep Spiking Neural Networks for Temporal Credit Assignment Problems. *IEEE Transactions on Neural Networks and Learning Systems*, 1–11.

Qin, L.; Yan, R.; and Tang, H. 2023. A Low Latency Adaptive Coding Spike Framework for Deep Reinforcement Learning. In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence*, 3049–3057.

Rao, A.; Plank, P.; Wild, A.; and Maass, W. 2022. A Long Short-Term Memory for AI Applications in Spike-based Neuromorphic Hardware. *Nature Machine Intelligence*, 4: 467–479.

Rashid, T.; Samvelyan, M.; de Witt, C. S.; Farquhar, G.; Foerster, J. N.; and Whiteson, S. 2018. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*, 4292–4301.

Rathi, N.; and Roy, K. 2023. DIET-SNN: A Low-Latency Spiking Neural Network With Direct Input Encoding and Leakage and Threshold Optimization. *IEEE Transactions on Neural Networks and Learning Systems*, 34(6): 3174–3182.

Roy, K.; Jaiswal, A.; and Panda, P. 2019. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575: 607–617.

Samvelyan, M.; Rashid, T.; de Witt, C. S.; Farquhar, G.; Nardelli, N.; Rudner, T. G. J.; Hung, C.; Torr, P. H. S.; Foerster, J. N.; and Whiteson, S. 2019. The StarCraft Multi-Agent Challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2186–2188.

Saravanan, M.; Kumar, P. S.; Dey, K.; Gaddamidi, S.; and Kumar, A. R. 2021. Exploring Spiking Neural Networks in Single and Multi-agent RL Methods. In *2021 International Conference on Rebooting Computing (ICRC)*, 88–98.

Seung, H. S. 2003. Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. *Neuron*, 40: 1063–1073.

Sun, Y.; Zeng, Y.; and Li, Y. 2022. Solving the spike feature information vanishing problem in spiking deep Q network with potential based normalization. *Frontiers in Neuroscience*, 16.

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

Tan, W.; Patel, D.; and Kozma, R. 2021. Strategy and Benchmark for Converting Deep Q-Networks to Event-Driven Spiking Neural Networks. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 9816–9824.

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

Tang, G.; Kumar, N.; Yoo, R.; and Michmizos, K. P. 2020. Deep Reinforcement Learning with Population-Coded Spiking Neural Network for Continuous Control. In *Proceedings of the 2020 Conference on Robot Learning*, 2016–2029.

Urbanczik, R.; and Senn, W. 2009. Reinforcement learning in populations of spiking neurons. *Nature Neuroscience*, 3: 250–252.

Wang, Z.; Jiang, R.; Lian, S.; Yan, R.; and Tang, H. 2023. Adaptive Smoothing Gradient Learning for Spiking Neural Networks. In *Proceedings of the 40th International Conference on Machine Learning*, 35798–35816.

Williams, R. J. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8: 229–256.

Wu, J.; Chua, Y.; Zhang, M.; Li, G.; Li, H.; and Tan, K. C. 2023. A Tandem Learning Rule for Effective Training and Rapid Inference of Deep Spiking Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 34(1): 446–460.

Wu, Y.; Deng, L.; Li, G.; Zhu, J.; and Shi, L. 2018. Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks. *Frontiers in Neuroscience*, 12.

Yin, B.; Corradi, F.; and Bohté, S. M. 2020. Effective and Efficient Computation with Multiple-timescale Spiking Recurrent Neural Networks. In *Proceedings of the International Conference on Neuromorphic Systems*, 1:1–1:8.

Zhang, D.; Zhang, T.; Jia, S.; and Xu, B. 2022. Multiscale Dynamic Coding improved Spiking Actor Network for Reinforcement Learning. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, 59–67.

Zhang, L.; Cao, J.; Zhang, Y.; Zhou, B.; and Feng, S. 2021. Distilling Neuron Spike with High Temperature in Reinforcement Learning Agents. *CoRR*, abs/2108.10078.

Zhang, W.-H.; Wu, S.; Josić, K.; and Doiron, B. 2023. Sampling-based Bayesian inference in recurrent circuits of stochastic spiking neurons. *Nature Communications*, 14: 7074.