Population-coding and Dynamic-neurons improved Spiking Actor Network for Reinforcement Learning

Duzhen Zhang\textsuperscript{a,b,*}, Tielin Zhang\textsuperscript{a,b,**}, Shuncheng Jia\textsuperscript{a,b}, Xiang Cheng\textsuperscript{a,b}, Bo Xu\textsuperscript{a,b,c,**}

\textsuperscript{a}Institute of Automation, Chinese Academy of Sciences (CAS), Beijing, China
\textsuperscript{b}University of the Chinese Academy of Sciences, Beijing, China
\textsuperscript{c}Center for Excellence in Brain Science and Intelligence Technology, CAS, China

Abstract

With the Deep Neural Networks (DNNs) as a powerful function approximator, Deep Reinforcement Learning (DRL) has been excellently demonstrated on robotic control tasks. Compared to DNNs with vanilla artificial neurons, the biologically plausible Spiking Neural Network (SNN) contains a diverse population of spiking neurons, making it naturally powerful on state representation with spatial and temporal information. Based on a hybrid learning framework, where a spike actor-network infers actions from states and a deep critic network evaluates the actor, we propose a Population-coding and Dynamic-neurons improved Spiking Actor Network (PDSAN) for efficient state representation from two different scales: input coding and neuronal coding. For input coding, we apply population coding with dynamically receptive fields to directly encode each input state component. For neuronal coding, we propose different types of dynamic-neurons (containing 1st-order and 2nd-order neuronal dynamics) to describe much more complex neuronal dynamics. Finally, the PDSAN is trained in conjunction with deep critic networks using the Twin Delayed Deep Deterministic policy gradient algorithm (TD3-PDSAN). Extensive experimental results show that our TD3-PDSAN model achieves better performance than state-of-the-art models on four OpenAI gym benchmark tasks. It is an im-

\*Duzhen Zhang and Tielin Zhang are co-first authors of this paper
\**Corresponding author

Email addresses: tielin.zhang@ia.ac.cn (Tielin Zhang), xubo@ia.ac.cn (Bo Xu)
portant attempt to improve RL with SNN towards the effective computation satisfying biological plausibility.

*Keywords:* Spiking neural networks, Reinforcement learning, Dynamic neuron, Population coding, Continuous control

1. Introduction

Reinforcement Learning (RL) occupies an interesting place in the world of machine learning algorithms [1], in which models interact with the environment in a trial-and-error manner and learn the optimal policy by maximizing accumulated rewards, so as to reaching excellent decision-making performances [2]. However, for traditional RL, it is a challenging problem to efficiently extract and represent features in complex high-dimensional state space. Deep Reinforcement Learning (DRL) has resolved this problem to some extent, by directly extracting features from high-dimensional raw input with the Deep Neural Networks (DNNs). Hence, DRL can be used to solve the decision-making problems of agents in complex tasks [3], such as recommendation systems [4] [5], games [6] [7], and robotic control [8] [9] [10].

Spiking Neural Network (SNN) is inspired by the biological brain, which is naturally the basic intelligent agent used for interactive reinforcement learning with the complex environment. Compared to DNNs with vanilla artificial neurons, SNN inherently transmits and computes information with dynamic spikes distributed over time [11]. The spike-based information encoding in SNN at both temporal and spatial dimensions will contribute to a more powerful state representation in RL [12] [13].

There are two main categories of information encoding (rate and temporal types) at the input coding scale of SNN. Rate coding uses the firing rate of spike trains in a time window to encode information, where input real numbers are converted into spike trains with a frequency proportional to the input value [14] [15], and temporal coding encodes information with the relative timing of individual spikes, where input values are usually converted into spike trains
with the precise time \[16, 17, 18, 19\]. Besides that, population coding is special in integrating these two types. For example, each neuron in a population can generate spike trains with precise time and also contain a relation with other neurons (e.g., Gaussian receptive field) for better information encoding at a global scale \[20, 21\].

For the neuronal coding scale of SNN, there are various types of spiking neurons \[22, 23\]. The integrate-and-fire (IF) neuron is the simplest neuron type. It fires when the membrane potential exceeds the firing threshold and the potential is then reset as a predefined resting membrane potential \[24\]. Another leaky integrate-and-fire (LIF) neuron allows the membrane potential to keep shrinking over time by introducing a leak factor \[22\]. They are commonly used as standard 1st-order neurons. Moreover, the Izhikevich neuron with 2nd-order equations of membrane potential is proposed, which can better represent the complex neuron dynamics, but requires some predefined hyper-parameters \[25\].

In this paper, based on a hybrid learning framework where a spike actor network infers actions from states and a deep critic network evaluates the actor, we propose a Population-coding and Dynamic-neurons improved Spiking Actor Network (PDSAN) for efficient state representation from two different scales: input coding outside the network and neuronal coding inside the network. For input coding, we apply population coding to input states where groups of learnable receptive fields are used to code each input component. The coded analog information is directly inputted into the network to improve computational efficiency and state representation capacity. For neuronal coding, different types of Dynamic Neurons (DNs) with 1st-order or higher-order dynamics of membrane potentials are proposed, and combined with population coding for stronger state representation capacity. Different from the predefined Izhikevich neurons, the dynamic neurons are self-learned from one of OpenAI gym \[26\] tasks (e.g., Ant-v3) and then extend to other similar tasks (e.g., HalfCheetah-v3, Walker2d-v3, and Hopper-v3) with the hypothesis that similar tasks are mostly possible share similar parameters. Finally, the proposed PDSAN integrates with the Twin Delayed Deep Deterministic policy gradient algorithm (TD3-PDSAN) \[27\] to learn
effective solutions for four continuous control tasks in the standard OpenAI gym \cite{26}. Compared to current state-of-the-art models, our proposed TD3-PDSAN model achieves better performances (rewards gained).

The main contributions of this paper can be concluded in the following parts:

- We combine the spatial coding and population coding, where each analog number in the input vector (state) is encoded as a group of analog numbers with learnable Gaussian receptive fields. We also test the differences in spatial coding and temporal coding (e.g., further encoding analog numbers into spike trains) and conclude that the spatial is relatively more efficient than other temporal codings.

- We construct a multi-layer spike actor network with dynamic neurons, containing 1st-order and higher-order neuronal dynamics for complex spatial and temporal information representation.

- With efficient state representation at both input and neuron scales, our proposed TD3-PDSAN model achieves new state-of-the-art performance on OpenAI gym benchmark tasks, including Ant-v3, HalfCheetah-v3, Walker2d-v3, and Hopper-v3.

2. Related Work

Recently, the literature has grown up around the theme of introducing SNNs in various RL algorithms \cite{28, 29, 30}. Some algorithms \cite{31} have extended the continuous Temporal Difference (TD) learning \cite{32} to the case of spiking neurons in an actor-critic network operating in continuous time. An RL with spiking neurons is implemented through two different synapse plasticities: stochastic and deterministic \cite{33}. These approaches are typically based on reward-modulated local plasticity rules that perform well in simple control tasks, but commonly fail in complex robotic control tasks due to limited optimization capability.

Some methods directly convert Deep Q-Networks (DQNs) \cite{6} to SNNs and achieve competitive scores on Atari games with discrete action space \cite{34, 35}.
However, these converted SNNs usually exhibit inferior performance to DNNs with the same structure [30]. Other methods utilize the backpropagation (BP) algorithm to train SNNs through substituting non-differential parts with a constant differential variable (approximate BP) [37, 38].

A hybrid learning framework is then proposed, trained by an approximate BP algorithm for mapless navigation of a mobile robot [39]. It contains two separate networks, where a spiking actor-network with basic LIF neurons infers actions from rate-coded states to represent the policy, and a deep critic network evaluates the actor by calculating action value. However, the rate-coded state has limited representation ability, which may affect the optimality of the policy and require a large number of timesteps for high performance at the expense of high inference latency and energy cost [40, 24]. A Population-coded Spiking Actor Network (PopSAN) with the same hybrid framework is designed to enhance state representation and achieves comparative performance on complex continuous control tasks [40]. Specifically, the input states with lower data dimensions are first transformed into the stimulation population coding of each value with the relative higher data dimensions. Then the computed stimulation codings are used to generate the neurons’ spike trains. This method improves the state representation capacity of the spike actor network mainly at input scale, but at the same time, decreases the computational efficiency.

By contrast, based on the hybrid learning framework, our proposed PDSAN further improves state representation capacity from two different scales: input coding outside the network and neuronal coding inside the network. At the input coding scale, unlike PopSAN, after applying population coding to input states, the coded analog information is inputted directly into the network for relatively high computational efficiency and representation capacity. At the neuronal coding scale, the dynamic neurons have the 1st-order or higher-order dynamics of membrane potentials to describe much more complex neuronal dynamics instead of LIF neurons with standard 1st-order dynamics. Our model achieves new state-of-the-art performance on complex continuous control tasks with efficient state representation at both input and neuron scales. Further-
more, there are other notable works on efficient coding for better information representation in SNNs \[41\] \[42\] \[43\].

3. Background

This section will introduce some basic RL theories and their related mathematical backgrounds. Some important algorithms that might contribute to a better understanding of our model are also introduced, including Deep Q-Networks (DQNs) \[6\], Deep Deterministic Policy Gradient algorithm (DDPG) \[44\], Twin Delayed Deep Deterministic policy gradient algorithm (TD3) \[27\], and also the hybrid learning framework \[39\] \[40\].

3.1. Reinforcement learning foundation

A standard reinforcement learning is commonly formalized as a Markov Decision Process (MDP), which is defined as a 5-tuple \((S, A, R, p, \rho_0, \gamma)\) where \(S\) is the state space, \(A\) is the action space, \(R: S \times A \times S \to \mathbb{R}\) is the reward function, \(p\) is the transition probability function, \(\rho_0\) is the initial state distribution and \(\gamma \in [0, 1]\) is the discount factor. At each time step \(t\), the agent is given a state \(s \in S\) and selects an action \(a \in A\) with respect to its policy \(\pi: S \to A\). In return, the agent receives a scalar reward \(r\) and the next state of the environment \(s'\). The process continues until the agent reaches the terminal state, and then the process restarts. The return is defined as the discounted sum of rewards \(R_t = \sum_{i=t}^{T} \gamma^{i-t}r(s_i, a_i)\) with the discount factor \(\gamma\) determining the priority of short term rewards. A trajectory \(\tau = (s_0, a_0, s_1, a_1, \ldots)\) is a sequence of states and actions, where \(s_0 \sim \rho_0\) and \(a_i \sim \pi\). A transition is a tuple \((s, a, r, s')\), where action \(a\) is performed at state \(s\), receiving next state \(s'\) and reward \(r\). The objective of reinforcement learning is to find an optimal policy that maximizes the cumulative discounted reward \(R_t\). The action value function \(Q(s, a) = \mathbb{E}_{\tau \sim \pi}[R(\tau)|s_0 = s, a_0 = a]\) measures the quality of an action after giving action \(a\) and state \(s\). Similarly, the state value function \(V(s) = \mathbb{E}_{\tau \sim \pi}[R(\tau)|s_0 = s]\) measures the quality of a specific state \(s\). Accurate estimation of the value can guide the algorithm to learn a better policy.
When the transition probability function is unknown, the action value can be recursively estimated by Bellman equation \[45\] that describes a fundamental relationship between the value of a state-action pair \((s, a)\) and the value of the subsequent state-action pair \((s', a')\):

\[
Q(s, a) = r + \gamma E_{s', a'}[Q(s', a')],
\]

(1)

where \(s' \sim p(\cdot|s, a)\) and \(a' \sim \pi(s)\).

For a large state space, the action value can be estimated with a differentiable function approximator \(Q(s, a; \theta)\), such as DNN with parameters \(\theta\). However, due to the property of generalization of DNN, there is a tendency to have a large variance. Moreover, function approximation error, model bias, and data noise might cause estimation bias.

3.2. DQNs

Given a state \(s\), DQNs exploit DNNs with parameter \(\theta\) to estimate the action value function \(Q(s, a; \theta)\). Two important tricks have been proposed, including experience replay and target network, to address the instability from the combination between DNNs and Q-learning, and thus dramatically improve the performance of DQNs \[6\]. The parameters of DQNs can be learned by minimizing the following loss function according to the Bellman equation (1):

\[
L(\theta) = E_{(s, a, r, s') \sim B}[(y - Q(s, a; \theta))^2],
\]

(2)

where \(y = r + \gamma \max_{a' \in A} Q(s', a'; \theta')\) is target action value computed by separated and frozen target network. The parameters of target network \(\theta'\) are periodically copied from the parameters of online learning network \(\theta\) to decouple correlation between online learning action value and target action value. And \(B\) is the replay buffer that stores the past transitions to reduce the correlation of sampled transitions.

For achieving better performance, plenty of remarkable methods to improve DQNs have been proposed, such as better exploration \[46\], recurrent neural network \[47\], and function regularization \[9\]. These methods learn a value...
function, known as a critic, and learn a policy, known as an actor, which can process continuous action space. Moreover, the actor is optimized by policy gradient algorithm [48].

3.3. DDPG

A Deterministic Policy Gradient method (DPG) is proposed to efficiently optimize the expected return and estimate value functions [49]. Similar to DQN [6], DDPG that combines DPG with DNNs is also proposed [44], which exploits the learned action-value function to optimize the policy.

We use $\phi, \phi', \theta, \theta'$ to denote the parameters of actor network, target actor network, critic network, and target critic network, respectively. The critic network is updated in the same way as the DQNs:

$$L(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{B}}[(y - Q(s,a;\theta))^2],$$

where $y = r + \gamma Q(s', \pi(s'; \phi'); \theta')$ is the target action value computed by the independent target networks. Based on critic network $\theta$, actor network is updated through the DPG [49]:

$$\nabla_{\phi} J(\phi) = \mathbb{E}_{s \sim p_{\pi}}[\nabla_a Q(s,a;\theta)|_{a=\pi(s;\phi)}] \nabla_{\phi} \pi(s;\phi)].$$

Different from DQNs, the parameters of target networks, $\phi', \theta'$, are periodically soft-updated from the parameters of online learning networks, $\phi, \theta$, to improve the stability of the learning process:

$$\theta' = \eta \theta + (1 - \eta) \theta', \quad \phi' = \eta \phi + (1 - \eta) \phi'.$$

where $\eta$ is a small enough constant.

3.4. TD3

TD3 is the state-of-the-art method that learns a deterministic policy. Based on DDPG, there are two main improvements in TD3 [27]. The first improvement is that TD3 takes the minimum value in a pair of target critics as the target action value to alleviate the overestimation phenomenon, called clipped double
Q-learning. The loss function of critic in TD3 is the same with DDPG [3], but
\[ y = r + \gamma \min_{i=1,2} Q'_i(s', \pi(s'; \phi'; \theta'_i)), \]
where \( Q'_1, Q'_2 \) represent two target critics with respect to two independent critics \( Q_1, Q_2 \). The second improvement is to delay actor update until critic network is updated after a fixed number of time steps \( d \), which decouples actor from critic and reduces function estimation error [27, 50].

3.5. Hybrid learning framework

Like its deep network counterpart, there are two independent networks in the hybrid learning framework [39, 40], where a spiking actor network represents the policy and a deep critic network evaluates the actor. The two networks in this framework can be trained jointly using approximate BP. Given a state \( s \), the spiking actor network generates an action \( a \), and the deep critic network estimates the associated action-value \( Q(s, a) \) (or state-value \( V(s) \)), which in turn optimizes the spiking actor network with a specified DRL algorithm. A spiking actor network is functionally equivalent to a deep actor network and can be integrated with any actor-critic based DRL algorithm [40], such as DDPG, TD3, and so on.

Figure 1: The overview of the architecture of TD3-PDSAN, including spiking actor networks (PDSAN) and deep critic networks.
4. Methods

The overview of our TD3-PDSAN model is presented in Figure 1. Our PDSAN is trained in conjunction with deep critic networks (i.e., a multi-layer fully-connected network) using the TD3 algorithm. During training, the PDSAN infers an action $a \in \mathbb{R}^m$ from a given state $s \in \mathbb{R}^n$, and the deep critic networks estimate the associated action-value $Q(s, a)$ to guide the PDSAN to learn a better policy. After training, the learned PDSAN can be applied to actual task scenarios to interact with the environment.

For efficient state representation, input coding outside the network and neuronal coding inside the network are proposed from different scales, and the overview of different state representation types is presented in Figure 2. In the input coding module of the PDSAN, each dimension of the state is encoded with population coding directly without additional rate coding and then fed into the multi-layer fully-connected SNNs. The DNs in the SNNs contain 2nd-order dynamic membrane potentials with up to two equilibrium points or 1st-order dynamic membrane potentials with up to one equilibrium point to describe complex neuronal dynamics. Similar to [40], the average firing rate is decoded into a corresponding action dimension with the population decoder.
4.1. Input coding

In this section, we introduce various types of input coding methods in SNNs. For a state $s \in \mathbb{R}^n$, we use these methods to generate input $I(t), t = 1, 2, ..., T_1$ for every timestep, where $T_1$ is the time window of the SNN.

4.1.1. Uniform coding (uni)

We first normalize the input $s$ and generate random number $Rand_i(t)$ with evenly distribution from 0 to 1, which has the same size as input at every time step. Then we compare every generated random number with its corresponding input data. If the generated random number is less than its input data, $I_i(t)$ is set to 1, or else it is set to 0, formulated as:

$$I_i(t) = \begin{cases} 1, & s_i > Rand_i(t); \\ 0, & \text{otherwise}. \end{cases}$$

(6)

where $Rand_i(t) \sim U(0, 1), i = 1, ..., n, t = 1, ..., T_1$, and $U$ is a uniform distribution.

4.1.2. Poisson coding (poi)

The Poisson random process with parameter $\lambda$ is used for the probability calculation of Poisson coding, formulated as:

$$P(k) = \begin{cases} \frac{e^{-\lambda} \lambda^k}{k!}, & k \in 1, 2, 3, ... \\ 0, & \text{otherwise}. \end{cases}$$

(7)

Consider that the Poisson process can be considered as the limit of a Bernoulli process, the normalized input $s$ containing probability can be used for drawing the binary random number as a bernoulli distribution. Then the $I_i(t)$ will draw a value 1 according to the $i^{th}$ probability value $s_i$ given in $s$, formulated as:

$$\begin{cases} P(I_i(t) = 1) = s_i \\ P(k) = \binom{N}{m} P^m I_i(t)=1 P^{N-m} I_i(t)=0 \end{cases}$$

(8)

where $i = 1, ..., n, t = 1, ..., T_1$. 11
4.1.3. Deterministic coding (det)

Similar to one-step soft-reset IF neurons, the normalized $s$ acts as the presynaptic inputs to the postsynaptic neurons [40], formulated as:

$$v(t) = v(t - 1) + s,$$

$$I_i(t) = 1 \& v_i(t) = v_i(t) - (1 - \delta), \text{ if } v_i(t) > 1 - \delta,$$

where $\delta$ is a small constant, $v$ is membrane voltage, $i = 1, ..., n$, $t = 1, ..., T_1$.

4.1.4. Population coding (pop)

For each dimension of state $s_i$, we create a population of neurons $E_i$ to encode it, where each neuron in $E_i$ has a learnable Gaussian receptive field $(\mu_{ij}, \sigma_{ij})$, $j = 1, ..., p$, and $p$ is the number of neurons in $E_i$. The pop is formulated as:

$$A_{E_{ij}} = e^{-\frac{(s_i - \mu_{ij})^2}{2\sigma_{ij}^2}},$$

$$A_{E_i} = (A_{E_{i1}}, A_{E_{i2}}, ..., A_{E_{ip}}) \in \mathbb{R}^p,$$

$$A_E = [A_{E_1}; A_{E_2}; ...; A_{E_n}] \in \mathbb{R}^{n \times p},$$

$$I(t) = A_E,$$

where $A_{E_{ij}}$ is the stimulation strength for each neuron in the population $E_i$, $[,]$ denotes concatenation and $j = 1, ..., p$, $i = 1, ..., n$, $t = 1, ..., T_1$.

A further analysis and comparison of all the input coding methods will be discussed in Section 5.4.

4.1.5. Population coding with rate

The population coding with rate contains two phases [40]: It first transforms state $s$ into stimulation strength $A_E$ by the original population coding; Then the computed $A_E$ is used to generate input trains $I(t)$ by the rate coding. It is formulated as:

$$I(t) = \text{Rate}(A_E),$$

where $\text{Rate}(\cdot)$ can be uniform coding (pop-uni), Poisson coding (pop-poi) or deterministic coding (pop-det), and $t = 1, ..., T_1$. 
4.2. DNs

In this section, we first introduce the traditional 1st-order neurons (e.g., LIF neurons) with a maximum one equilibrium point for Ordinary Differential Equation (ODE) of membrane potential, and then define the improved 2nd-order neurons. These neurons are all considered the DNs for the basic description of neuronal dynamics in SNNs. The procedure for constructing these DNs is also introduced in the following sections.

4.2.1. The traditional 1st-order neurons

The traditional 1st-order neurons in SNNs are the LIF neurons, which are the simplest abstraction of the Hodgkin–Huxley model. In order to show the basic equilibrium point characteristics of LIF neurons, here we give a simple definition of LIF neurons as the following description:

\[ \tau \frac{dV_{i,t}}{dt} = -V_{i,t} + I, \]  

where the dynamic membrane potential \( V_{i,t} \) is updated with the variable \( I \) (e.g., the input post-synaptic current). It is obvious to calculate the single equilibrium point of \( v_{i,t}^* \) at \( I \) within period time \( \tau \).

The number of equilibrium points will be the key to distinguish different levels of neuronal dynamics. The dynamic field of \( V_{i,t} \) is also shown in Figure 3 (a,b) with \( I = 5 \).

4.2.2. The designed 2nd-order neurons

The neurons with higher-order dynamics means that the number of equilibrium points for these neurons will be more than one. The definition of these neurons are shown as following formulations:

\[ \tau \frac{dV_{i,t}}{dt} = -V_{i,t}^N + I, \]  

where the \( N \) is the order of dynamics with \( N > 1 \). Hence, the dynamic membrane potential \( V_{i,t} \) will be attracted or nonstable at some points when we set ODE with 0. Figure 3 (c,d) show a diagram of one dynamic field of membrane
potential with $N = 2$. For equilibrium points, the period for reaching stable points will also cost time $\tau$.

For biological neurons, our previous work more focus on the 2nd-order dynamics with $N = 2$, as following formulation:

\[
\frac{dV_{j,t}}{dt} = V_{j,t}^2 - V_{j,t} - U_{j,t} + \sigma \left( \sum_{i=1}^{N} W_{i,j} I_{i,t} \right),
\]  

(18)

where $V_{j,t}^2$, $V_{j,t}^1$, and $V_{j,t}^0$ are membrane potentials with different dynamics, $U_{j,t}$ is a resistance item simulating hyper polarization. For simplicity, here, we formulate the basic unit as 1 instead of traditional units, such as mV and ms.

Besides membrane potential, some other implicit variables are also used for
the dynamic description of 2nd-order dynamics, shown as follows:

\[
\begin{cases} 
  \frac{dU_{j,t}}{dt} = \theta_a (\theta_b V_{j,t} - U_{j,t}) \\
  V_{j,t} = \theta_c \text{ if } (V_{j,t} > V_{th}) \\
  U_{j,t} = U_{j,t} + \theta_d \text{ if } (V_{j,t} > V_{th})
\end{cases}
\]

(19)

where the equilibrium point of \( V_{j,t} \) is decided by both \( U_{j,t} \) and input currents \( I_t = \sigma(\sum_{i=1}^N W_{i,j} I_{i,t}) \), \( I_{i,t} \) is the pre-synaptic input (from neuron \( i \)). \( W_{i,j} \) is the synaptic weight, \( \sigma(\cdot) \) is a sigmoid function for input normalization. The number and value of equilibrium points will be dynamically effected by parameters \( \theta_{a,b,c,d} \), which will be further identified during the learning procedure in the next section.

4.2.3. The procedure for constructing the DNs

The construction of different DNs is mainly based on the identification of some key parameters in dynamic neurons. As \( \theta_{a,b,c,d} \), for example, each setting of these four parameters describes one dynamic state of a spiking neuron. Hence, for SNN, three layers of neurons (with each layer contains hundreds of neurons) will be initialized from uniformly-distributed random parameters from zero to one, as shown in Figure 1.

These learnable parameters of \( \theta_{a,b,c,d} \), combining with other synaptic weights \( W_{i,j} \), will be tuned on one of the tasks with TD3-PDSAN algorithm. After learning, where most of the learnable variables reach the stable points, these parameters will be plotted and clustered with the k-means method to get the best center of \( \theta_{a,b,c,d} \) parameters. These four key parameters will be further used as the unified configuration of all dynamic neurons for all tasks.

4.3. The forward propagation of PDSAN and the learning procedure of TD3-PDSAN

The forward propagation of PDSAN and the learning procedure of TD3-PDSAN are shown in Algorithm 1 and Algorithm 2 (in the section 7 Appendix), respectively.
Algorithm 1: Forward propagation through PDSAN

Initialize coding means $\mu$ and standard deviations $\sigma$ for all population encoders;

Randomly initialize weight matrices $W$ and biases $b$ for each SNN layer;

Load the best dynamic parameters of DNs $\theta^* = (\theta^*_a, \theta^*_b, \theta^*_c, \theta^*_d)$ (pre-learning from a task);

Randomly initialize decoding weight vectors $W_d$ and bias $b_d$ for each action dimension;

Initialize the current decay factor $d_c$ and firing threshold $v_{th}$;

$n$-dimensional state, $s$;

Inputs from populations generated by the input coding module:

$A_E = [..., \exp \left( \frac{(s_i - \mu_i)^2}{2\sigma_i^2} \right), ..., i = 1, ..., n]$;

for $t = 1, ..., T_1$ do
  Inputs at timestep $t$: $o^{(t)(0)} = I(t) = A_E$;
  for $l=1,...,L$ do
    Update DNs in layer $l$ at timestep $t$ based on spikes from layer $l-1$:
    $c^{(t)(l)} = d_c \cdot c^{(t-1)(l)} + W^{(l)} o^{(t)(l-1)} + b^{(l)}$;
    $v^{(t)(l)} = v^{(t-1)(l)} \cdot (1 - o^{(t-1)(l)}) + o^{(t-1)(l)} \cdot \theta^*_c$;
    $u^{(t)(l)} = u^{(t-1)(l)} + o^{(t-1)(l)} \cdot \theta^*_d$;
    $Y(u^{(t)(l)}, v^{(t)(l)}) \begin{cases} v_{\text{delta}} = v^{(t)(l)} \cdot v^{(t)(l)} - v^{(t)(l)} - u^{(t)(l)} + c^{(t)(l)} \\ u_{\text{delta}} = \theta^*_a \cdot (\theta^*_b \cdot v^{(t)(l)} - u^{(t)(l)}) \end{cases}$;
    $v^{(t)(l)} = v^{(t)(l)} + v_{\text{delta}}$;
    $u^{(t)(l)} = u^{(t)(l)} + u_{\text{delta}}$;
    $o^{(t)(l)} = (v^{(t)(l)} > v_{th})$;
  end
  Sum up the output spikes: $sc = \sum_{t=1}^{T_1} o^{(t)(L)}$;
  Compute the average firing rate: $fr = sc / T_1$;
  Divide $fr$ into $m$ output populations: $\{fr^{(j)}\}, j = 1, ..., m$;
  Generate $m$-dimensional action $a$ by the population decoder:
  $a_j = W_d^{(j)} \cdot fr^{(j)} + b_d^{(j)}, j = 1, ..., m$;
4.4. Training PDSAN with approximate BP

Our previous work has discussed different methods for tuning multi-layer SNNs, including approximate BP [41], equilibrium balancing [51, 52], Hopfield-like tuning [53], and plasticity rules inspired from biologies [54]. In this paper, we select the approximate BP for its efficient learning of large numbers of parameters, and at the same time, remaining the key relationship between these parameters. The key feature of approximate BP is converting standard BP into a piecewise version of BP, where the non-differential parts of spiking neurons can be replaced with a predefined gradient, shown as equation (23).

Here, we analyze the step-by-step flow of the gradients during the training of PDSAN. \( \nabla_a J \) is the gradient of the loss for the computed action, which is used to optimize the parameters of PDSAN. The parameters for each output population \( j, j \in 1, \ldots, m \) are updated as follows:

\[
\nabla f^{(j)} \cdot J = \nabla a_j \cdot W^{(j)}_d, \quad (20)
\]
\[
\nabla W^{(j)}_d \cdot J = \nabla f^{(j)} \cdot f^{(j)}, \quad (21)
\]
\[
\nabla b^{(j)}_d \cdot J = \nabla f^{(j)}, \quad (22)
\]

where \( f^{(j)} \) is the average firing rate and \( W^{(j)}_d, b^{(j)}_d \) is the decoding parameters for each output population.

The parameters of SNN are updated using the approximate BP, where we use the rectangular function equation to approximate the gradient of a spike.

\[
z(v) = \begin{cases} 
1 & \text{if } |v - v_{th}| < w, \\
0 & \text{otherwise},
\end{cases} \quad (23)
\]

where \( z \) is the pseudo-gradient, \( v \) is membrane voltage, \( v_{th} \) is firing threshold and \( w \) is the threshold window for passing the gradient.

For each timestep, \( t < T_1 \), we describe the flow of the gradients through the
SNN. At the output population layer \( L \), we have:

\[
\nabla_{SC} J = \frac{1}{T_1} \cdot \nabla_{fr} J,
\]

\[
\nabla_{o^{(L)}(L)} J = \nabla_{SC} J,
\]

where \( SC \) is the summation of output spikes over time window \( T_1 \) and \( o^{(L)}(L) \) is the output spikes of layer \( L \)-th at time \( t \).

Then for each layer, \( l = L \) down to 1:

\[
\nabla_{v^{(l)}(l)} J = \nabla_{v^{(l)}(l)} o^{(l)}(l) \cdot \nabla_{o^{(l)}(l)} J + (1 - o^{(l)}(l)) \cdot \nabla_{v^{(l+1)}(l)} J,
\]

\[
\nabla_{v^{(l)}(l)} J = \sum_{t=1}^{T_1} o^{(l)}(l) \cdot \nabla_{o^{(l)}(l)} J,
\]

\[
\nabla_{o^{(l)}(l-1)} J = W^{(l)} \cdot \nabla_{o^{(l)}(l)} J,
\]

where \( c \) is the current, \( d_c \) is the current decay factor.

\[
\nabla_{v^{(l)}(l)} o^{(l)}(l) = \begin{cases} 
 z(v^{(l)}(l)) & \text{LIF neurons} \\
 z(v^{(l)}(l)) \cdot \nabla_{v^{(l)}(l)} Y(u^{(l)}(l), v^{(l)}(l)) & \text{DNs}
\end{cases}
\]

where the forward calculation process of \( Y(u^{(l)}(l), v^{(l)}(l)) \) is in Algorithm 1.

When \( t = T_1 \), by collecting the gradients backpropagated from all the timesteps, the gradient of the loss with respect to the SNN parameters for each layer \( l \) can be calculated:

\[
\nabla_{W^{(l)}} J = \sum_{t=1}^{T_1} o^{(l)(l-1)} \cdot \nabla_{o^{(l)}(l)} J,
\]

\[
\nabla_{b^{(l)}} J = \sum_{t=1}^{T_1} \nabla_{o^{(l)}(l)} J.
\]

Finally, we computed the gradient of the loss \( J \) with respect to the parameters of each input population \( i, i \in 1, ..., n \):

\[
\nabla_{A^{(i)}} J = \sum_{t=1}^{T_1} \nabla_{A^{(i)(o)}} J,
\]

\[
\nabla_{\mu^{(i)}} J = \nabla_{A^{(i)}} J \cdot A^{(i)} \cdot \frac{s_i - \mu^{(i)}}{\sigma^{(i)2}},
\]

\[
\nabla_{\sigma^{(i)}} J = \nabla_{A^{(i)}} J \cdot A^{(i)} \cdot \frac{(s_i - \mu^{(i)})^2}{\sigma^{(i)3}}.
\]

We update all the parameters of PDSAN after every \( T_1 \) timesteps.
5. Experiments

![Figure 4: Example OpenAI gym tasks (a) Ant-v3, (b) HalfCheetah-v3, (c) Walker2d-v3, (d) Hopper-v3.](image)

Table 1: The details of the tasks.

| Task          | State Dimension (n) | Action Dimension (m) | Goal                                                                 |
|---------------|---------------------|----------------------|----------------------------------------------------------------------|
| Ant-v3        | 111                 | 8                    | Make a four-legged creature walk forward as fast as possible          |
| HalfCheetah-v3| 17                  | 6                    | Make a 2D cheetah robot run as fast as possible                      |
| Walker2d-v3   | 17                  | 6                    | Make a 2D bipedal robot walk forward as fast as possible             |
| Hopper-v3     | 11                  | 3                    | Make a 2D one-legged robot hop forward as fast as possible           |

To evaluate our model, we measured its performance on four continuous control tasks from the OpenAI gym (Figure 4) [26]. The details of these tasks were shown in Table 1. The goals of our experiments were as follows:

* To learn the dynamic parameters of DNs from a task and analyze the membrane-potential dynamics of DNs (Section 5.2);
* To demonstrate the integration of PDSAN with the TD3 algorithm by reaching benchmark performance of our method against the corresponding deep actor networks and PopSAN (Section 5.3);
5.1. Implement details

Due to recent concerns in reproducibility [55], all our experiments were reported over 10 random seeds of the network initialization and gym simulator. Each task was run for 1 million steps and evaluated every 10k steps, where each evaluation reported the average reward over 10 episodes with no exploration noise, and each episode lasted for a maximum of 1000 execution steps.

We compared our TD3-PDSAN (integrate PDSAN with TD3 algorithm) against TD3 (integrate deep actor network with TD3 algorithm), TD3-Pop (integrate population coding and deep actor network with TD3 algorithm, it has the same amount of parameters as TD3-PDSAN) and TD3-PopSAN (integrate PopSAN with TD3 algorithm), where the hyper-parameter configurations of deep actor network and PopSAN were the same as the ones used in [40]. Unless explicitly stated, PDSAN and PopSAN training used the same hyper-parameters as the deep actor network. The configuration of hyper-parameters for these models was set as follows:

(1) TD3:

Actor network was (256, relu, 256, relu, action dim $m$, tanh); Critic network was (256, relu, 256, relu, 1, linear); Actor learning rate was $10^{-3}$; Critic learning rate was $10^{-3}$; Reward discount factor was $\gamma = 0.99$; Soft target update factor was $\eta = 0.005$; Maximum length of replay buffer was $T = 10^6$; Gaussian exploration noise was $\sigma = 0.1, \tilde{\sigma} = 0.2$; Noise clip was $c = 0.5$; Mini-batch size was $N = 100$; Policy delay factor was $d = 2$. 
(2) TD3-Pop:
Actor network was (Population Encoder, 256, relu, 256, relu, Population Decoder, action dim \( m \), tanh); Input population size for single state dimension was \( p = 10 \); Input coding used population coding (pop for all tasks); Other configurations are the same as TD3.

(3) TD3-PopSAN:
PopSAN was (Population Encoder, 256, LIF, 256, LIF, Population Decoder, action dim \( m \), tanh), where the current decay factor, voltage factor and firing threshold of LIF neurons were 0.5, 0.75, and 0.5, respectively; Input population size for single state dimension was \( p = 10 \); Time window was \( T_1 = 5 \); PopSAN learning rate was \( 10^{-4} \); Input coding used pop-det (HalfCheetah-v3 & Ant-v3) and pop-poi (Hopper-v3 & Walker2d-v3).

(4) TD3-PDSAN:
PDSAN used (Population Encoder, 256, DNs, 256, DNs, Population Decoder, action dim \( m \), tanh), where the current decay factor and firing threshold of MDNs were both 0.5; Input population size for single state dimension was \( p = 10 \); Time window was \( T_1 = 5 \); PDSAN learning rate was \( 10^{-4} \); Input coding used population coding (pop for all tasks).

5.2. Learn and analyze DNs

![Training curve in Ant-v3](image1)
![Candidate (\( \theta_a, \theta_b \)) from Ant-v3](image2)
![Candidate (\( \theta_c, \theta_d \)) from Ant-v3](image3)

Figure 5: The selection process of the best dynamic parameters \( \theta_{a,b,c,d} \). (a) The evaluation reward curve of Ant-v3. (b) The clustering of the candidate parameters of \( \theta_a \) and \( \theta_b \) learned from Ant-v3, where the center of the clustering was selected as the best set (labelled as a red triangle). (c) Similar with that of (b) but for candidate parameters of \( \theta_c \) and \( \theta_d \).
We selected Ant-v3 as the basic source task for the pre-learning of 2nd-order DNs with TD3-PDSAN, and then trained all parameters together (containing both synaptic weights and also the dynamic parameters of DNs) with BP (or approximate BP for SAN), as shown in Figure 5(a).

As shown in Figure 5(b-c), then we obtained a clustering center of parameters $\theta_a$ and $\theta_b$ in Figure 5(b) and parameters $\theta_c$ and $\theta_d$ in Figure 5(c), respectively. Here we set $k = 1$ in k-means for simplicity. The best dynamic parameters of DNs were $\theta^* = (\theta^*_a = -0.172, \theta^*_b = 0.529, \theta^*_c = 0.021, \theta^*_d = 0.132)$. Then the $\theta^*$ would be further used as the unified configuration of all dynamic neurons for all tasks in the following experience.

![Figure 6](image)

**Figure 6**: The neuronal dynamics on membrane potential of 1st-order LIF neurons (a) and 2nd-order DNs (b), where the red lines represented dynamic membrane potential $V$, the green lines were dynamic resistance item $U$, the blue lines were simulated input and the yellow lines were the values of equilibrium points for membrane potentials.

The neuronal dynamics for different explicit (e.g., the membrane potential $V$ and stimulated input $I$) and implicit variables (e.g., resistance item $U$ and equilibrium point values) were shown in Figure 6.

For the standard LIF neuron in Figure 6(a), the membrane potential was positively proportional to the neuron input. For example, for the sin-like input with the value range from -1 to 1, the dynamic $V$ was dynamically integrated until reaching the firing threshold $V_{th}$ only with a strong positive, or else, decay accordingly with weak-positive or negative stimulus.
Unlike LIF neurons, the DNs showed a higher complexity with an additional implicit $U$, making the dynamical changing of equilibrium points different. The little differences of $U$ would cause a big update of $V$ according to the definition of DNs, especially when the parameter $\theta_b$ was small in Equation (19). Hence, the DNs would not only show similar firing patterns with the positive strong stimulus but also exhibit a sparse firing with the weak-positive and negative stimulus, instead of stopping firing like LIF neurons. This result showed the better dynamic representation of DNs compared to LIF neurons.

5.3. Benchmarking PDSAN against deep actor networks and PopSAN

![Figure 7: The comparison of average rewards for different algorithms. (a) The performance of TD3, TD3-Pop, TD3-PopSAN and TD3-PDSAN during training on Ant-V3 gym task. (b,c,d) The performances of these three algorithms on HalfCheetah-v3, Walker2d-v3, and Hopper-v3, respectively, where our proposed TD3-PDSAN got the best performance. The shaded region represented half a standard deviation of the average evaluation over 10 seeds and the curves were smoothed for clarity.](image)

We compared the performance of our TD3-PDSAN with TD3, TD3-Pop and TD3-PopSAN. As Figure 7 shown, our algorithm achieved the best performance across all the tested tasks, which showed the effectiveness of our proposed algorithm for continuous control tasks. In addition, TD3-Pop has not brought any obvious advantage for most of the four tasks (except HalfCheetah-v3) compared to TD3. Further analysis in Figure 8 shows that spiking actor networks with population coding achieved significant performance improvements compared to that without population coding. Hence, in conclusion, population coding contributes to spiking actor networks but brings no obvious advantage to deep actor networks. This may be because the ”over-parameterized” network may be hard
to train in some of the tasks when population coding was combined with a deep actor network.

5.4. The comparison of various input coding methods

![Comparison of coding methods](image)

Figure 8: Comprehensive comparison of the impact of various input coding methods on performance. The pop achieved the best overall performance.

We comprehensively compared the impact of various input coding methods on performance while keeping the neuronal coding method fixed to DNs. As Figure 8 shown, the performance of the rate coding method alone (poi) was far inferior to population coding based methods (pop-uni, pop-poi, pop-det, pop) on all four tasks. This could be because the rate coding methods had an inherent limitation on the representation capacity of individual neurons. As for the population coding based methods, pop achieved the best performance on tasks ANT-V3, HalfCheetah-v3 and Walker2d-v3, and was comparable to other population coding based methods on tasks Hopper-v3. The performance of the other three population coding based methods varied greatly depending on the specific task. It seemed to be more efficient to directly use the analog value of the state after population coding as the network input, without further using rate coding to encode analog value into spike trains. Besides, we evaluated pop with different input population sizes per state dimension: $p = 2, 5, 10$. Figure 11 (in the Appendix 7) showed that the performance on Ant-v3 task deteriorated when reducing the size of input population.

5.5. The representation capabilities of DNs

We tested the constructed DNs on all four tasks and compared them with the LIF neurons while keeping the input coding method fixed to Population coding
Figure 9: The DNs almost consistently performed better than LIF neurons on all four tasks.

As shown in Figure 9, the DNs reached a better performance than LIF neurons on all tested tasks, including the source task (where the dynamic parameters of DNs were learned, i.e., Ant-v3) and other similar tasks (i.e., HalfCheetah-v3, Walker2d-v3, and Hopper-v3). This result initially verified the hypothesis that similar tasks were mostly possible share similar parameters, i.e., the dynamic parameters of DNs learned from a task could be generalized to other similar tasks. Although there was no rigorous theoretical proof, we had done many experiments to verify this hypothesis further. We collected a set of spatial datasets, including MNIST, Fashion-MNIST, NETtalk, and Cifar-10, and temporal datasets, including TIDigits and TIMIT. We learned a set of dynamic neurons from MNIST and TIDigits, called spatial dynamic neurons and temporal dynamic neurons, respectively. Then we tested the performance of different dynamic neurons on different tasks, and the results were summarized in Table 2.

We could conclude from Table 2 that the spatial dynamic neurons were more powerful on spatial tasks, while the temporal dynamic neurons performed better on temporal tasks. This result was consistent with our previous hypothesis that when tasks had similar attributes and backgrounds, the dynamic parameters learned from one of the tasks could be generalized to other tasks and bring performance improvements to other tasks (e.g., spatial dynamic neurons applied to spatial tasks). When tasks were of different types, the dynamic parameters learned from one of the tasks had reduced the performance of other tasks (e.g., spatial dynamic neurons applied to temporal tasks).
Table 2: Performance of different dynamic neurons on different tasks, with unit of (%) for following accuracies. The bold numbers are the best accuracy on specific tasks.

| Tasks      | Spatial dynamic neurons (learned from MNIST) | Temporal dynamic neurons (learned from TIDigits) |
|------------|---------------------------------------------|-------------------------------------------------|
| MNIST      | 98.67 ± 0.05                                | 94.87 ± 0.08                                    |
| Fashion-MNIST | 90.50 ± 0.10                               | 85.37 ± 0.06                                    |
| NETtalk    | 91.46 ± 0.36                                | 87.15 ± 0.20                                    |
| Cifar-10   | 54.30 ± 0.16                                | 47.26 ± 0.15                                    |
| TIDigits   | 65.76 ± 0.79                                | 78.00 ± 0.44                                    |
| TIMIT      | 87.61 ± 2.39                                | 91.22 ± 1.13                                    |

Figure 10: The neuronal spiking activity after training on HalfCheetah-v3 task, where x-coordinate is the simulation time (ms) and y-coordinate is the neuron index (id) selected at random. (a) Spiking activity of 10 out of 512 sample LIF neurons. (b) Spiking activity of 10 out of 512 sample DNs.

For the performance gap between LIF neurons and DNs, it was significant to understand the nature of these neurons. Unlike LIF neurons with standard 1st-order dynamics, the DNs contained the 1st-order and higher-order dynamics
of membrane potentials and showed a higher complexity, which contributed to a more powerful state representation. In addition, we also recorded the spiking activity of LIF neurons and DNs after training on HalfCheetah-v3 task in Figure 10. It could be observed that the spikes for LIF neurons were sparser, and the spike count or fire rate of LIF neurons was smaller than that of DNs, which might be one of the reasons for their performance gap.

6. Conclusion

The state representation is important in the research of both SNN and RL. This paper integrates Population-coding at network input and the DNs coding inner networks towards an efficient Spiking Actor Network (PDSAN) that performs even better or comparable to other state-of-the-art DNN-based algorithms on some benchmark Open-AI gym tasks.

The DNs make the neurons with higher computational complexity, showing more complicated membrane potential dynamics than simple LIF neurons. We think the increment of complicity at neuron scale will contribute more to that at network scales. This characteristic might also show an advantage on the energy-efficient computation. Besides, spikes generated by DNs in PDSAN make the computational cost between neurons lower than their counterpart DNNs. We think biology’s inspiration will give us more hints towards better algorithms with faster learning convergence, lower energy cost, stronger adaptability, higher robust computation and better interpretability.

Acknowledgment

This study is supported by the National Key R&D Program of China (Grant No. 2020AAA0104305), the National Natural Science Foundation of China (Grant No. 61806195), the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDB32070100, XDA27010404), and the Beijing Brain Science Project (Z181100001518006).
References

[1] L. P. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement Learning: A Survey, J. Artif. Intell. Res. 4 (1996) 237–285.

[2] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.

[3] S. S. Mousavi, M. Schukat, E. Howley, Deep reinforcement learning: an overview, in: Proceedings of SAI Intelligent Systems Conference, Springer, 2016, pp. 426–440.

[4] L. Zou, L. Xia, Z. Ding, J. Song, W. Liu, D. Yin, Reinforcement Learning to Optimize Long-term User Engagement in Recommender Systems, in: A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, G. Karypis (Eds.), Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019, ACM, 2019, pp. 2810–2818.

[5] R. Warlop, A. Lazaric, J. Mary, Fighting Boredom in Recommender Systems with Linear Reinforcement Learning, in: S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, 2018, pp. 1764–1773.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nat. 518 (7540) (2015) 529–533.

[7] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan,
M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnovets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, Ç. Gülçehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. P. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, D. Silver, Grandmaster level in StarCraft II using multi-agent reinforcement learning, Nat. 575 (7782) (2019) 350–354.

[8] Y. Duan, X. Chen, R. Houthooft, J. Schulman, P. Abbeel, Benchmarking Deep Reinforcement Learning for Continuous Control, in: M. Balcan, K. Q. Weinberger (Eds.), Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016, Vol. 48 of JMLR Workshop and Conference Proceedings, JMLR.org, 2016, pp. 1329–1338.

[9] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, in: J. G. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, Vol. 80 of Proceedings of Machine Learning Research, PMLR, 2018, pp. 1856–1865.

[10] D. Han, K. Daya, J. Tani, Variational Recurrent Models for Solving Partially Observable Control Tasks, in: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net, 2020.

[11] W. Maass, Networks of spiking neurons: The third generation of neural network models, Neural Networks 10 (9) (1997) 1659–1671.

[12] E. Painkras, L. A. Plana, J. D. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, S. B. Furber, SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation, IEEE J. Solid State Circuits 48 (8) (2013) 1943–1953.
[13] M. Davies, N. Srinivasa, T. Lin, G. N. Chinya, Y. Cao, S. H. Choday, G. D. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, H. Wang, Loihi: A Neuromorphic Manycore Processor with On-Chip Learning, IEEE Micro 38 (1) (2018) 82–99.

[14] P. U. Diehl, M. Cook, Unsupervised learning of digit recognition using spike-timing-dependent plasticity, Frontiers Comput. Neurosci. 9 (2015) 99.

[15] X. Cheng, Y. Hao, J. Xu, B. Xu, LISNN: Improving Spiking Neural Networks with Lateral Interactions for Robust Object Recognition, in: C. Bessiere (Ed.), Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, ijcai.org, 2020, pp. 1519–1525.

[16] R. Güttig, To spike, or when to spike?, Current opinion in neurobiology 25 (2014) 134–139.

[17] I. M. Comsa, T. Fischbacher, K. Potempa, A. Gesmundo, L. Versari, J. Alakuijala, Temporal Coding in Spiking Neural Networks with Alpha Synaptic Function, in: 2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020, IEEE, 2020, pp. 8529–8533.

[18] L. Zhang, S. Zhou, T. Zhi, Z. Du, Y. Chen, TDSNN: From Deep Neural Networks to Deep Spike Neural Networks with Temporal-Coding, in: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, AAAI Press, 2019, pp. 1319–1326.

[19] A. G. Sboev, D. Vlasov, R. B. Rybka, A. Serenko, Spiking neural network reinforcement learning method based on temporal coding and STDP, in:
A. V. Samsonovich, C. Lebiere (Eds.), Postproceedings of the 9th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA 2018 (Ninth Annual Meeting of the BICA Society), August 22-24, 2018, Prague, Czech Republic, Vol. 145 of Procedia Computer Science, Elsevier, 2018, pp. 458–463.

[20] S. M. Bohte, H. L. Poutré, J. N. Kok, Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks, IEEE Trans. Neural Networks 13 (2) (2002) 426–435.

[21] A. P. Georgopoulos, A. B. Schwartz, R. E. Kettner, Neuronal population coding of movement direction, Science 233 (4771) (1986) 1416–1419.

[22] W. Gerstner, W. M. Kistler, Spiking neuron models: Single neurons, populations, plasticity, Cambridge university press, 2002.

[23] H. C. Tuckwell, Introduction to theoretical neurobiology: volume 2, non-linear and stochastic theories, no. 8, Cambridge University Press, 1988.

[24] N. Rathi, K. Roy, DIET-SNN: Direct Input Encoding With Leakage and Threshold Optimization in Deep Spiking Neural Networks, CoRR abs/2008.03658. arXiv:2008.03658

[25] E. M. Izhikevich, Simple model of spiking neurons, IEEE Trans. Neural Networks 14 (6) (2003) 1569–1572.

[26] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI Gym, CoRR abs/1606.01540. arXiv:1606.01540

[27] S. Fujimoto, H. van Hoof, D. Meger, Addressing Function Approximation Error in Actor-Critic Methods, in: J. G. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, Vol. 80 of Proceedings of Machine Learning Research, PMLR, 2018, pp. 1582–1591.
[28] R. V. Florian, Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity, Neural computation 19 (6) (2007) 1468–1502.

[29] W. Potjans, A. Morrison, M. Diesmann, A Spiking Neural Network Model of an Actor-Critic Learning Agent, Neural Comput. 21 (2) (2009) 301–339.

[30] M. J. O’Brien, N. Srinivasa, A Spiking Neural Model for Stable Reinforcement of Synapses Based on Multiple Distal Rewards, Neural Comput. 25 (1) (2013) 123–156.

[31] N. Frémaux, H. Sprekeler, W. Gerstner, Reinforcement Learning Using a Continuous Time Actor-Critic Framework with Spiking Neurons, PLoS Comput. Biol. 9 (4).

[32] K. Doya, Reinforcement Learning in Continuous Time and Space, Neural Comput. 12 (1) (2000) 219–245.

[33] M. Yuan, X. Wu, R. Yan, H. Tang, Reinforcement Learning in Spiking Neural Networks with Stochastic and Deterministic Synapses, Neural Comput. 31 (12) (2019) 2368–2389.

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

[35] W. Tan, D. Patel, R. Kozma, Strategy and Benchmark for Converting Deep Q-Networks to Event-Driven Spiking Neural Networks, CoRR abs/2009.14456. arXiv:2009.14456

[36] N. Rathi, G. Srinivasan, P. Panda, K. Roy, Enabling Deep Spiking Neural Networks with Hybrid Conversion and Spike Timing Dependent Backpropagation, in: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net, 2020.
[37] S. Woźniak, A. Pantazi, T. Bohnstingl, E. Eleftheriou, Deep learning incorporating biologically inspired neural dynamics and in-memory computing, Nature Machine Intelligence 2 (6) (2020) 325–336.

[38] F. Zenke, S. Ganguli, SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks, Neural Comput. 30 (6).

[39] G. Tang, N. Kumar, 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, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021, IEEE, 2020, pp. 6090–6097.

[40] G. Tang, N. Kumar, R. Yoo, K. P. Michmizos, Deep Reinforcement Learning with Population-Coded Spiking Neural Network for Continuous Control, CoRR abs/2010.09635. \texttt{arXiv:2010.09635}.

[41] X. Cheng, T. Zhang, S. Jia, B. Xu, Finite Meta-Dynamic Neurons in Spiking Neural Networks for Spatio-temporal Learning, CoRR abs/2010.03140. \texttt{arXiv:2010.03140}.

[42] N. K. Kasabov, Time-space, spiking neural networks and brain-inspired artificial intelligence, Springer, 2019.

[43] K. Kumarasinghe, N. Kasabov, D. Taylor, Brain-inspired spiking neural networks for decoding and understanding muscle activity and kinematics from electroencephalography signals during hand movements, Scientific reports 11 (1) (2021) 1–15.

[44] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. S. andF Daan Wierstra, Continuous control with deep reinforcement learning, in: Y. Bengio, Y. LeCun (Eds.), 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, 2016.
[45] R. Bellman, R. Kalaba, On the role of dynamic programming in statistical communication theory, IRE Trans. Inf. Theory 3 (3) (1957) 197–203.

[46] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, R. Munos, Unifying Count-Based Exploration and Intrinsic Motivation, in: D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, R. Garnett (Eds.), Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, 2016, pp. 1471–1479.

[47] M. J. Hausknecht, P. Stone, Deep Recurrent Q-Learning for Partially Observable MDPs, in: 2015 AAAI Fall Symposia, Arlington, Virginia, USA, November 12-14, 2015, AAAI Press, 2015, pp. 29–37.

[48] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, Policy Gradient Methods for Reinforcement Learning with Function Approximation, in: S. A. Solla, T. K. Leen, K. Müller (Eds.), Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999], The MIT Press, 1999, pp. 1057–1063.

[49] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M. A. Riedmiller, Deterministic Policy Gradient Algorithms, in: Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014, Vol. 32 of JMLR Workshop and Conference Proceedings, JMLR.org, 2014, pp. 387–395.

[50] Q. He, X. Hou, WD3: taming the estimation bias in deep reinforcement learning, in: 32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020, IEEE, 2020, pp. 391–398.

[51] M. Shi, T. Zhang, Y. Zeng, A Curiosity-Based Learning Method for Spiking Neural Networks, Frontiers in Computational Neuroscience 14 (2020) 7.
7. Appendix
Algorithm 2: TD3-PDSAN

Initialize PDSAN $\pi$, and critic networks $Q_1, Q_2$ with random parameters $\phi, \theta_1, \theta_2$

Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$

Initialize replay buffer $B$

Initialize $d, \sigma, \bar{\sigma}, \eta, c$ and total steps $T$

Reset the environment and receive initial state $s$

for $t = 1$ to $T$ do

Select action with exploration noise $a \sim \pi(s; \phi) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma)$ and receive reward $r$ and new state $s'$

Store transition tuple $(s, a, r, s')$ in $B$

Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $B$

$\tilde{a} \leftarrow \pi(s'; \phi') + \epsilon, \epsilon \sim \text{clip}(\mathcal{N}(0, \bar{\sigma}), -c, c)$

$y \leftarrow r + \gamma \min_{i=1,2} Q(s', \tilde{a}; \theta'_i)$

Update critics $\theta_i \leftarrow \min_{\theta} N^{-1} \sum (y - Q(s, a; \theta_i))^2$

if $t \mod d$ then

Update $\phi$ by deterministic policy gradient:

$\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q(s, a; \theta_1)|_{a = \pi(s; \phi)} \nabla_\phi \pi(s; \phi)$

Soft update target networks:

$\theta'_i \leftarrow \eta \theta_i + (1 - \eta) \theta'_i$

$\phi' \leftarrow \eta \phi + (1 - \eta) \phi'$

end

end
Figure 11: The performance comparisons of different $p$ hyperparameter configurations in population coding (pop).