electronics

MDPI

Article

Spiking Neural-Networks-Based Data-Driven Control

Yuxiang Liu [1] and Wei Pan [2,*]

[1] Computer Engineering Laboratory, Faculty of Electronic Engineering, Mathematics and Computer Science, Delft University of Technology, Building 36, Mekelweg 4, 2628 CD Delft, The Netherlands
[2] Department of Cognitive Robotics, Faculty of Mechanical, Maritime and Materials Engineering, Delft University of Technology, Building 34, Mekelweg 2, 2628 CD Delft, The Netherlands
* Correspondence: wei.pan@tudelft.nl

Abstract: Machine learning can be effectively applied in control loops to make optimal control decisions robustly. There is increasing interest in using spiking neural networks (SNNs) as the apparatus for machine learning in control engineering because SNNs can potentially offer high energy efficiency, and new SNN-enabling neuromorphic hardware is being rapidly developed. A defining characteristic of control problems is that environmental reactions and delayed rewards must be considered. Although reinforcement learning (RL) provides the fundamental mechanisms to address such problems, implementing these mechanisms in SNN learning has been underexplored. Previously, spike-timing-dependent plasticity learning schemes (STDP) modulated by factors of temporal difference (TD-STDP) or reward (R-STDP) have been proposed for RL with SNN. Here, we designed and implemented an SNN controller to explore and compare these two schemes by considering cart-pole balancing as a representative example. Although the TD-based learning rules are very general, the resulting model exhibits rather slow convergence, producing noisy and imperfect results even after prolonged training. We show that by integrating the understanding of the dynamics of the environment into the reward function of R-STDP, a robust SNN-based controller can be learned much more efficiently than TD-STDP.

Keywords: spiking neural network; reinforcement learning; control

Citation: Liu, Y.; Pan, W. Spiking Neural-Networks-Based Data-Driven Control. *Electronics* **2023**, *12*, 310. https://doi.org/10.3390/electronics12020310

Academic Editors: Zhan Li, Zhang Chen and Yiyong Sun

Received: 30 November 2022
Revised: 27 December 2022
Accepted: 27 December 2022
Published: 7 January 2023

1. Introduction

A traditional control loop can be conceptualized into several essential blocks [1], including the internal and external observation blocks, the control block, and the (optional) dynamic model block. The internal and external observation blocks describe the (past and) current states of the system and the environment. These states are used by the control block to determine which future states the system should reach to achieve certain control goals and what actions should be taken to bring the system into these future states. In model-based control [2], the decision making of the control block is assisted by the dynamic model block that predicts the system's behavior (e.g., through simulations of the physical system). In contras, in model-free control [3], no dynamic model of the system is required.

For many control systems, blocks that perform one or more of the above functions can be advantageously implemented with machine learning (ML). This field has experienced rapid developments in recent decades and is having substantial impact in various disciplines of science and engineering [4]. In control engineering, ML approaches have been extensively investigated for tasks, such as industrial process control [5], autonomous vehicle maneuvering [6], and robotics operation [7].

ML can be applied to preprocess complex sensory data before these data are used to make control decisions [8]. Moreover, supervised learning (one type of ML) [9,10] can be applied to produce a dynamic model [11], which can be used in place of a physics-based model if the physics model is overly simplified, difficult to derive, or too time-consuming to simulate. At the control loop's core, ML can achieve robust nonlinear mapping from

high-dimensional state variables to optimal control decisions. Reinforcement learning (RL) [12], a subfield of machine learning, has been widely recognized as targeting tasks that generally fall within the scope of control engineering [13].

Most commonly, the ML approaches used for control tasks have been based on artificial neural networks (ANNs), especially deep ANNs (i.e., ANNs with many hidden layers) [14,15]. To a large extent, the versatility and effectiveness of ANNs can be attributed to their hierarchical organization of relatively simple computational units into dedicated network structures, which gained inspiration from intricately connected neuron networks in biological brains [16]. However, despite such a structural similarity between ANNs and biological neural networks, the computations performed by the nodes of ANNs (the artificial "neurons") are fundamentally different from the "computations" performed by the biological neurons. While the nodes in ANNs usually use numerical values as inputs and outputs, biological neurons use time-dependent signal series (i.e., spike trains) as inputs and outputs [17].

A typical neuron cell has many dendrites and an axon connected to neurons upstream and downstream through microstructures called synapses [18]. Interconnected neurons use spikes to communicate with each other [17]. Over a while, a series of spikes (a spike train) generated by a pre-synaptic neuron can modulate the voltage level of a postsynaptic neuron. When the latter voltage is above a threshold value, the postsynaptic neuron generates its spike (that is, it is firing), and its voltage level drops. In a spiking neuron network (SNN), each neuron will receive spike trains from the upstream neurons connecting to its input synapses, produce its spike train, and pass its spike train to the downstream neurons connected with its output synapses [19].

Vreeken et al. proposed that computational neural networks can be divided into three generations [20]: the first and second generations use artificial neurons that accept real-valued inputs and generate outputs of binary (first generation) or continuous (second generation) values, while the third generation uses neurons that consider time-dependent signal series (i.e., spike trains) as input and output, mimicking the signaling between actual neuron cells in biological brains [17].

One of the most attractive characteristics of SNNs is that they can potentially offer much higher energy efficiency than conventional ANNs [21]. The neural activity in an SNN is event-driven because a neuron is only active when it receives or fires a spike, while it can be idle when there is no event. This is different from an artificial neuron in an ANN, which must always be active. Energy-efficient SNN computations are actively pursued through the development of SNN-based ML algorithms [22] and neuromorphic hardware that allows on-chip SNNs [23,24].

This paper investigates the use of SNN instead of ANN in control problems. In particular, we are interested in designing SNNs to carry out the task of making optimal control decisions. Since the most widely used ML approach for such tasks is reinforcement learning (RL), in what follows, we will first present a brief introduction to the basic techniques of RL, followed by an overview of existing learning methods for SNN.

Using the classical cart-pole game as an example, we constructed a simple yet effective SNN to solve this control problem with delayed rewards. We examine the performance of two learning methods adapted from spike-timing-dependent plasticity (STDP). The learning method that introduces the temporal difference signal into the training process presents a general framework for tackling various control problems. The other method that employs the reward signal in the training process exploits our prior knowledge of a specific control problem and therefore trades its generality for the efficiency of solving that specific control problem.

2. Preliminary

2.1. A Brief Introduction to Reinforcement Learning (RL)

In a typical RL scheme, an agent (a learner is often noted as an agent in RL) is in one of a range of possible states, noted as $s \in S$, where S is the state space. In each state, the

agent can take an action a from a predefined set A, which denotes the action space. The agent's action can potentially change the agent's current state. At each state, the agent may receive a certain amount of reward. The agent's goal is to choose a proper action at each given state so that the total reward the agent receives from the states it goes through will eventually be maximized.

The rule by which the agent chooses its action can be noted as the policy π. For example, π can correspond to the probability that the agent will take action a when in state s, that is, $\pi(s, a) = P(a|s)$ for $s \in S$ and $a \in A$. The goal of RL is to find the optimal policy for every state.

Assuming that the environment evolves in consecutive steps during learning, at a certain step n, the agent detects that the environment is at a state s_n and chooses an action a_n (according to its current policy). We define the Q value function as the averaged future total reward the agent can receive by following the policy π at all future states, namely,

$$Q_\pi(s_n, a_n) = E_\pi(G|s = s_n, a = a_n), \tag{1}$$

where G represents the total future rewards or gains, and E_π represents averaging over all possible evolutionary trajectories of future state sequences given the policy π.

Now, we consider the Q value function that corresponds to the optimal policy π_{opt} (i.e., the policy that maximizes the Q values) and note this function as Q_{opt}. Because the total future reward of step n is the sum of the reward for the state at step $n + 1$ and the total future reward of step $n + 1$, and also because the optimal policy at step $n + 1$ would be to take action with the maximum future gain after step $n + 1$, Q_{opt} should satisfy the following Bellman optimal equations, which Bellman proposed as the basis for solving the RL problem via a dynamic programming algorithm [25],

$$Q_{opt}(s, a) = R(s) + \Sigma_{s' \in S} P^a_{ss'} max_{a' \in A} Q_{opt}(s', a'), \tag{2}$$

where $P^a_{ss'}$ refers to the transition probability $P(s_{n+1} = s'|s_n = s, a_n = a)$.

To solve the Bellman equations, one can start with some initial Q value functions and update iteratively. In this process, we can define the deviations of the current Q value function from the Bellman equations as temporal differences, that is,

$$TD_Q = R(s) + \Sigma_{s' \in S} P^a_{ss'} max_{a' \in A} Q(s', a') - Q(s, a) \tag{3}$$

The optimal Q value function can be obtained by iteratively using the temporal differences to update the Q value function,

$$Q_{new}(s, a) = Q_{old}(s, a) + \gamma TD_Q, \tag{4}$$

in which γ is the learning rate.

The RL algorithm that derives the optimal policy by learning the Q value function is known as Q-Learning. When the state space and the action space are discrete, the Q value function is a table (i.e., Q-table) of Q values for all state–action pairs. Guided by the Q-table, the agent can choose the appropriate action based on its state. Its inventor Watkins proved that the Q learning algorithm based on Equation (4) can converge if the Q values are represented in a lookup table [26].

2.2. An Overview of Learning Methods for SNN

While the temporal dynamics of spiking neurons can be well-simulated by models such as the Leaky Integrate and Fire (LIF) Model [27–29], it is not immediately clear how to train networks composed of such neurons for diverse ML tasks [30], including RL. Unlike ANNs that use differentiable real-valued signals and can thus be trained with the well-known backpropagation algorithm [31], SNNs use discrete and nondifferentiable spike train signals, making training based on straightforward backpropagation unfeasible.

It is an active field of research to develop appropriate algorithms to train SNNs for different applications [19,32]. One way to train an SNN is first to train an ANN with the desired functionality and then convert the ANN into an SNN. This is also known as shadow training [19]. The rules for the conversion or replacement of ANN neurons by spiking neurons are to match the activation functions of the ANN nodes with the firing rates of the spiking neurons [33] or the timing of the spikes emitted by the spiking neurons [34]. Although the conversion approach has been shown to produce SNNs with an accuracy comparable to deep ANNs for certain image recognition tasks, the approach has several drawbacks. Training a complex ANN is time-consuming and converting an ANN to an SNN requires a rather long simulation time. Moreover, the approach usually leads to large and deep SNNs, with performances worse than the source ANNs since the conversion processes lead to no exact replications of but only approximations to the original ANNs [32].

Another type of SNN training algorithm enables backpropagation or gradient descent training by using differentiable approximations of the spike function for derivative estimation [35,36]. The Back Propagation Through Time or BPTT method [35] converts the neural network to an equivalent sequence of computations and applies the backpropagation framework to it. The SpikeProp method uses the fact that time is continuous and calculates the derivative of the spike timing concerning the weight [36]. Backpropagation-based training has produced SNNs that can solve small-scale image recognition problems well.

The above learning algorithms for SNN echo the learning methods for ANN. They do not correspond to how learning occurs in a biological brain. Neuroscience studies have long revealed synaptic plasticity, i.e., changes in the connection strengths of synapses, as a fundamental mechanism underlying biological learning. A highly interesting SNN learning algorithm uses a biological learning mechanism called spike-timing-dependent plasticity (STDP) of synapses [37].

STDP is a form of Hebbian plasticity that follows the rule first postulated by Hebb as "cells that fire together wire together" [38]. In other words, it is based on the correlated firing activity of presynaptic and postsynaptic neurons to decide whether to change the strength of connectivity of a synapse. The core idea behind these learning rules is to build synaptic connections to represent causal relationships. As exhibited by its name, STDP has precisely defined relationships between spike timing and changes in connection strengths [39,40]. Biologically, STDP is supported by the experimental finding that the co-activation of the pre-and post-synaptic neurons can lead to two types of plasticity or synaptic changes: long-term potentiation (LTP) and long-term depression (LTD) [41]. LTP refers to strengthening the synaptic connection when the pre-synaptic neuron fires within a short period before the firing of the post-synaptic neuron. In contrast, LTD refers to weakening the connection when firing events occur in reverse order. Equation (5) presents a simple mathematical model for STDP with LTP and LTD,

$$\Delta w = \begin{cases} A_+ exp(-\frac{s}{\tau_+}), & s > 0 \\ A_- exp(\frac{s}{\tau_-}), & s <= 0, \end{cases} \tag{5}$$

where Δw is the change in the strength of connection or synaptic weight, s is the time difference between postsynaptic and presynaptic spikes, and A_+ and A_- are, respectively, the coefficients for potentiation (weight increase) and depression (weight decrease). The time windows for LTP and LTD are given by τ_+ and τ_-, respectively [42].

The STDP learning mechanism provides a possible basis for training SNNs to make control decisions in a control loop. This can be achieved by introducing an additional modulating factor for the SNN synaptic plasticity process [43,44]. The modulating factor can be determined from the environment's reactions to the control decisions made according to the output of the SNN so that the finally trained SNN can produce control decisions that lead to optimal outcomes. Neuroscience studies have shown that in biological brains, dopamine is one of the most important neuromodulators in such feedback- or reward-based learning processes [45]. That is, the dopamine level represents the amount of rewards

that guide the learning process by modulating STDP [46]. This biological phenomenon can be simulated by extending the STDP model in formula 5 to the following modulated STDP formula,

$$\Delta w = M * w_{eligibility}, \quad (6)$$

where M represents the effects of neuromodulators, while $w_{eligibility}$ selects synapses for reward-modulated plasticity according to conventional STDP rules [43].

Several previous studies have investigated the application of the above idea of modulated STDP learning rules to various ML problems, including the making of control decisions [44,47]. Florian et al. introduced the reward-modulated STDP, or R-STDP algorithm [44], in which the modulating factor came from the deviations of the actual and expected SNN output. In that study, the algorithm's effectiveness was illustrated with the example of using SNN to solve the XOR problem and to output predefined spike trains. We note that these examples were essentially regression problems. The so-called "rewards" were derived from the regression errors, but not from a reaction of an environment impacted by a control decision (determined from the outcome of the SNN). In other words, the rewards for these problems were instantaneous but not delayed, as in typical control problems.

Frémaux et al. proposed using temporal difference (TD) in reinforcement learning (RL) as the modulating factor for STDP [47]. This TD-STDP learning rule was applied to train actor-critic-type reinforcement learning (RL) controllers [12,13]. In such a controller, an SNN subnetwork serving as the critic provides estimations of the "values" of the environmental states. During training, the temporal change of the values estimated by the critic was used to derive the modulating factor for STDP. The analysis in [47] illustrated the potential of using SNN to implement general RL strategies for control tasks.

Despite these previous studies, STDP-based SNN learning for control tasks remains an underexplored research problem. As mentioned above, the examples did not include typical control tasks with delayed rewards in the original study that examined R-STDP [44]. Moreover, there is a lack of studies that compare the relative advantages and drawbacks of different schemes, such as R-STDP and TD-STDP, for the same specific control task. In this paper, we explore SNN with STDP learning modulated with the reward factor for control tasks by considering the cart-pole balancing problem [48] as a representative example. We have designed and implemented a novel SNN controller trained by the TD-STDP rules similar to those proposed in ref. [47]. Based on the RL Q learning strategy [26], the SNN presented here has a much simpler general network architecture than the actor–critic architecture of ref. [47]. Although the TD-STDP learning rule is very general and does not require any specific assumption about the dynamics of the environment (i.e., the target to be controlled), it leads to rather slow convergence, producing somewhat imperfect controllers even after prolonged training. We have also examined R-STDP learning. We show that by integrating the understanding of the specific dynamics of the environment into the design of the reward function, a more robust SNN-based controller can be learned much more efficiently by R-STDP than by TD-STDP.

2.3. The Cart-Pole Environment

The cart-pole environment is one of the classic control problems that has been widely applied for the performance evaluation of control algorithms. Several studies on SNN have also chosen the cart-pole control problem as their test bench [49–51]. This problem could be stated as the following: given a cart with a pole attached to it through a fixed joint (i.e., the pole can only rotate in a plane perpendicular to the horizontal surface), the cart can move on a flat surface with no friction. At each step of the game, the control agent should choose one of two actions, pull right or pull left. The action will affect the state of the cart pole. The controller's goal is to keep the pole in approximately upright positions without falling for as long as possible. The values that the agent can observe are:

- The position of the cart: x,

- The velocity of the cart: $v = \frac{dx}{dt}$,
- The pole angle: θ,
- The angular velocity of the pole: $\omega = \frac{d\theta}{dt}$.

Here, right is defined as the positive direction for these values.

In this paper, the cart-pole model environment implemented in the gym library [52] is applied. This environment terminates its simulation if one of these three situations occurs:

- The pole drops when the absolute value of the angle of the pole is greater than 12°.
- The cart slides out of the edge of the animation display (i.e., the cart position is out of the range between -2.4 to 2.4).
- The simulation steps are larger than 200 (or other customized values).

By definition, the first two termination conditions correspond to the reach of some failure states of the environment, while the third termination condition indicates the successful completion of the task.

3. The TD-STDP SNN

3.1. The Overall Workflow of the Program

The overall framework of our program that uses TD-STDP learning to solve the cart-pole control problem is presented in Figure 1. The entire program can be divided into two main parts; one is the cart-pole environment and the other is the SNN-based controller. Each training epoch comprises a consecutive simulation of the cart-pole steps from the start until one of the finishing conditions is met. At each cart-pole step, the environment simulator and the SNN simulator are run sequentially. After a cart-pole simulation step, the SNN simulator takes the current state of the cart pole as its input (see below), runs for a fixed period, and produces its output, based on which an action to be applied to the cart-pole environment will be chosen. After taking action, the cart-pole environment evolves to the next step. The whole process is iterated until the simulation of the environment ends. Then, the cart-pole environment is reset and started from a new initial state to perform the next training epoch.

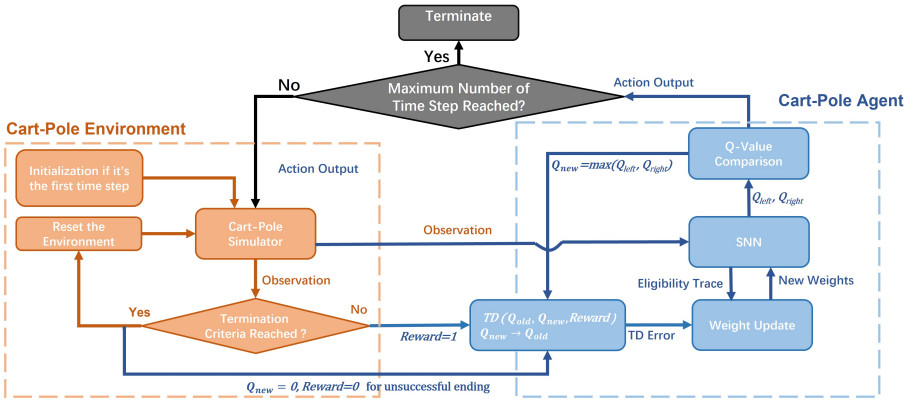


Figure 1. A general workflow of the training process using TD-STDP.

3.2. The Architecture of the SNN

The overall structure of the SNN is illustrated in Figure 2. There are only two layers of neurons. The first is the input layer, which comprises some input neurons. The second is the output layer, which comprises two groups of neurons, each corresponding to one possible choice of action. Every input neuron is connected to every output neuron through a synapse.

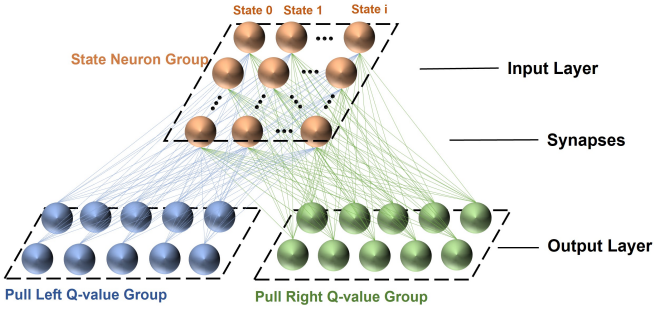


Figure 2. The overall structure of the SNN using TD-STDP.

3.3. The Input Neurons

The spikes generated by the input neurons code the observed state of the cart pole. First, an observed variable of the cart pole is mapped to an integer index using the following formulae.

$$id_{obs} = \begin{cases} 0, & obs \leq obs_{min} \\ floor(\frac{x - x_{min}}{\Delta x}), & obs_{min} < obs < obs_{max} \\ N_{states,obs} - 1, & obs \geq obs_{max} \end{cases} \tag{7}$$

$$N_{states,obs} = ceil(\frac{x_{max} - x_{min}}{\Delta x}), \tag{8}$$

where the variable obs corresponds to one of the observed variables x, v, θ, or ω; $[obs_{min}, obs_{max}]$ defines the region of values for evenly divided bins; Δx is the width of the bins; and $N_{states,obs}$ is the total number of bins or discrete states for the variable obs.

Any possible state of the cart pole is described by a unique combination of the four integers $(id_x, id_v, id_\theta, id_\omega)$. The total number of possible states is $N_{states,total} = N_{states,x} * N_{states,v} * N_{states,\theta} * N_{states,\omega}$.

The total number of SNN input neurons is defined to be $N_{states,total} * n_{input}$, so that a group of n_{input} neurons represents each possible state. When a given state of the cart pole is passed to the SNN, only the n_{input} neurons representing that particular state will fire spikes. Meanwhile, all other input neurons will remain inactive. This encoding technique is inspired by methods for encoding categorical variables for ML problems and is generally known as one-hot encoding [53].

3.4. The Output Neurons

An LIF model describes the dynamics of each output neuron with adaptation in Equation (9) [54].

$$\tau_m \frac{dV}{dt} = g_e(E_e - V) + E_l - V, \tag{9}$$

in which V is the membrane potential, E_l is the resting potential, E_e is a high value voltage, τ_m is a time constant for the membrane potential, and g_e is a dimensionless quantity representing the effects of the upstream spikes received through the input synapses. Without receiving spikes from the input synapses, the dynamics of g_e is described by

$$\tau_g \frac{dg_e}{dt} = -g_e, \tag{10}$$

in which τ_e is a time constant for the lasting effect of the input spikes. When an input spike from a synapse i connects to the neuron, g_e changes instantly according to the number of parameters.

$$g_e = g_e + w_i, \tag{11}$$

in which w_i is the weight of the synapse i. The effects of spikes received by different synapses are simply accumulated.

3.5. Q-Learning by SNN

The output of the SNN is interpreted as the (scaled) Q values of corresponding actions given the particular input state, namely,

$$Q(s,a) = scale * n_{spikes}(s,a), \tag{12}$$

in which $n_{spikes}(s,a)$ refers to the number of spikes fired by neurons in the output group associated with action a when only the input neurons that encode the state s have been firing during a period of SNN simulation.

Note that this definition of the output of the SNN as the (scaled) Q value that depends both on the state and the action is somewhat different from the SNN value network of ref. [47]. There, the output from a single group of neurons provided the values of only the states without further discrimination of different actions.

Using separated groups of output neurons for different actions, we do not need an extra actor module as in ref. [47] to evaluate the actions. Instead, the action is determined by the same groups of neurons that estimate the Q values.

To determine the TD error of the current SNN for the cart-pole step n, we run the SNN with the input state s_n for a while, obtain the values of $Q(s_n,a)$, choose and carry out the action a_n, and obtain the next state of the cart pole s_{n+1}. We then run the SNN with the input state s_{n+1} to obtain $Q(s_{n+1},a)$. Following Equation (3), the TD error for step n is computed as

$$TD_n = TD(s_n,a_n) = \gamma max_a Q(s_{n+1},a) + R_{n+1} - Q(s_n,a_n). \tag{13}$$

Here, we have included a "discount factor" γ so that the rewards of only a finite number of future steps effectively contribute to the current Q value.

The formula above with $R_{n+1} = 1$ is applied only when the state at step $n+1$ does not correspond to a failure state of the cart pole. When s_{n+1} corresponds to a failure state (and the simulation of the environment will terminate), both $Q(s_{n+1},a)$ and R_{n+1} should be zero (because there will be no current or future reward). Then, the TD error for step n is computed as

$$TD_n = -Q(s_n,a_n). \tag{14}$$

As shown in Figure 1, the TD error is used to update the weights of the synapses connecting the input state neurons to the output neurons.

3.6. Determining Eligibility of the Synapses

Given the current state of the cart pole, the SNN is simulated for a fixed number of SNN time steps (note that these are not the cart-pole steps). The SNN simulation produces two outcomes. The first is the estimated Q values described above. The second outcome includes the eligibility traces of the synapses for updating the synapses' weights. To determine the eligibility for each synapse (for simplicity, we will omit the index of the synapse from the following formulations) based on the relative timing of pre-and post-synaptic spikes, we first define the following pre-synaptic activity and post-synaptic activity traces,

$$A_{pre}(t) = \sum_{k \in input\ spikes} \xi(t-t_k) e^{-\frac{t-t_k}{\tau_{pre}}}, \tag{15}$$

$$A_{post}(t) = \sum_{k \in output\ spikes} \xi(t-t_k) e^{-\frac{t-t_k}{\tau_{post}}}, \tag{16}$$

in which t_k represents the time of spike k, and the Heaviside step function $\xi(t - t_k)$ is applied to select spikes that were received (for A_{pre}) or fired (for A_{post}) before the time t, i.e.,

$$\xi(t - t_k) = \begin{cases} 1, & t - t_k \geq 0 \\ 0, & otherwise. \end{cases} \tag{17}$$

A_{pre} is not negligible only for a short time (compared to the time constant t_{pre}) after receiving a presynaptic spike. Similarly, A_{post} is not negligible, only not long after the firing of a post-synaptic spike. Then, the eligibility of the synapse according to the STDP rule can be determined as

$$w_{trace} = \Delta_{pre} \int O(t) A_{pre}(t) dt - \Delta_{post} \int I(t) A_{post}(t) dt, \tag{18}$$

where Δ_{pre} and Δ_{post} are parameters that affect learning rates. In the above equation, the function $I(t)$ represents the time series of input spikes on that synapse, while the function $O(t)$ represents the time series of the output spikes, that is,

$$I(t) = \sum_{k \in input\ spikes} \delta(t - t_k), \tag{19}$$

$$O(t) = \sum_{k \in output\ spikes} \delta(t - t_k). \tag{20}$$

In Equation (18), the first term corresponds to long-term potentiation (LTP) of the synapses because it makes a positive contribution to w_{trace} only when an output spike is generated shortly after receiving a presynaptic spike (thus A_{pre} is non-negligible); the second term corresponds to long-term depression (LTD), as it makes a negative contribution to w_{trace} if an input spike is received at the synapses shortly after the post-synaptic neuron has fired (thus A_{post} is not negligible).

3.7. Learning the Synaptic Weights by TD-STDP

After the determination of TD_n, the weights of the synapses that connect to the output groups corresponding to an action a_n are updated using

$$w_{new} = w_{old} + \beta * TD_n * w_{trace,n}, \tag{21}$$

where β is a constant that adjusts the rate of the weight change, and $w_{trace,n}$ is the eligibility trace calculated using Equation (18) at the nth cart-pole step.

3.8. Exploration and Exploitation in Training

For the agent to accumulate experience in various cart-pole states, the agent should be encouraged to explore various possible states during the initial training phase of the SNN. This is achieved by adding a stochastic mechanism to choose the action. Specifically, for the first 100 epochs, the agent chooses an action randomly at every cart-pole step. After that, the agent randomly chooses an action with a probability of $P_{explore}$, the value of which starts from 1.0 and is scaled down at the beginning of each new epoch by a factor of $\alpha = 0.99$. With a probability of $1 - P_{explore}$, the agent chooses the action according to

$$P(a_n = a) \propto e^{(Q_a / \delta_{Q_0})}. \tag{22}$$

With the above definition, the agent's action would be deterministic only when the difference between the Q values of the two actions provided by the SNN is significant (relative to δ_{Q_0}, which is chosen to be 0.1).

4. The R-STDP SNN

4.1. The Differences between the R-STDP and the TD-STDP Programs

The overall framework of the R-STDP program shown in Figure 3 is similar to the overall framework of the TD-STDP program. The main differences between the two programs come from the different interpretations of the output of the SNN (i.e., the number of spikes fired by each of the two groups of neurons in the output layer). In the TD-STDP SNN, the output represents the scaled Q values for different actions of the corresponding input state. Thus, the relative output changes upon changes in the input state have meaning. In the R-STDP SNN, the output is (by definition) interpreted or used as simple numerical metrics for choosing the action (here, the action associated with the output group that fires the most spikes is preferred). Thus, when there are changes in the input state, the corresponding changes in the output of the R-STDP-trained SNN are not concerned. This leads to a different weight-updating scheme in the R-STDP program.

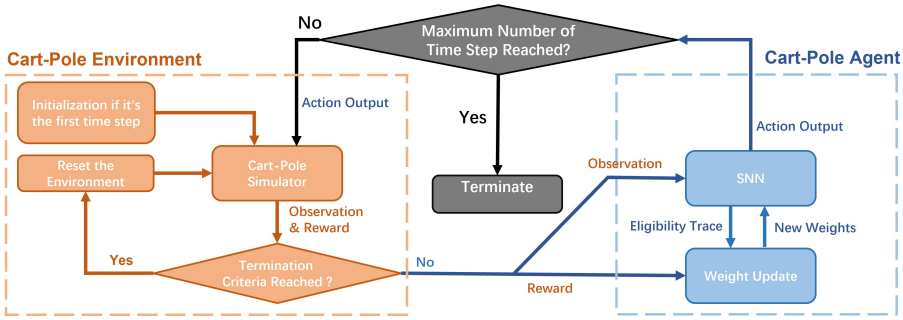


Figure 3. A general workflow of the training process using R-STDP.

4.2. Updating Synapse Weights with Delayed Reward

After the action (chosen on the basis of the output of the SNN) has been taken, a new state of the cart pole is returned. A reward value of R is calculated from both the previous state and the new state of the cart pole (see below). The weights of the SNN synapses are updated using the reward and eligibility factors computed from the previous SNN run. When the action leads to a positive reward, the synapses connections that would increase the probability of the SNN output of the action taken should be strengthened. In contrast, the synapse connections that would increase the probability of the SNN producing the opposite action should be weakened. The opposite should apply when the action taken leads to a negative reward. This reasoning leads to the following updating rules. For every synapse connected to the output neuron associated with the action taken, the weights increase according to the action.

$$w_{new} = w_{old} + R(s_n, s_{n+1}) * w_{trace,n}, \tag{23}$$

while for every synapse connected to the output neuron associated with the opposite of the taken action,

$$w_{new} = w_{old} - R(s_n, s_{n+1}) * w_{trace,n}. \tag{24}$$

4.3. Reward Function Designs

Intuitively, the reward function should reflect the objective of the control task, which is to keep the pole in the upward direction. As this goal should be realized by the actions that are chosen according to and carried out upon the given states of the cart pole, an action should be rewarded positively if it causes the state of the cart pole to change towards a new state that favors achieving the control goal. Based on this intuition, we designed and tested the three reward functions of the SNN controller. From Reward Function 1 to Reward Function 3, more sensory information is encoded into the reward function; therefore, the agent's actual tendency to keep the pole upright is more accurately pictured. In the following formulations, the subscript *old* corresponds to the cart-pole step n in

Equations (23) and (24), while the subscript *new* denotes the cart-pole step $n+1$ in the same equations.

Reward Function 1: This reward function is defined to return one when the angle of the pole is in the range where the pole does not fall, and the cart is still within the range of the animation display and otherwise returns zero. That is,

$$R_1 = \begin{cases} 1, & \textit{The simulation is not terminated} \\ 0, & \textit{otherwise.} \end{cases} \tag{25}$$

Reward Function 2: This reward function is defined as

$$R_2(\omega_{old}, \omega_{new}) = \begin{cases} 1, & \omega_{old} * \omega_{new} < 0 \textit{ or} \\ & |\omega_{old}| > |\omega_{new}| \\ -1, & \textit{otherwise.} \end{cases} \tag{26}$$

As shown in Figure 4a, this reward is determined by comparing the current angular velocity ω_{new} with the angular velocity observed in the previous cart-pole step ω_{old}. If a significant change in angular velocity has been detected, the action has reversed the previous moving trend of the pole, and thus a reward of one is assigned to this kind of action. Suppose that there is no significant change in angular velocity. In that case, the shrinking of the absolute value of the angular velocity itself also indicates that the pole is slowing down its rotation. Therefore, the action is also encouraged with a reward of one. In other situations, the reward value of -1 is used to punish the action taken for not contributing to reverse or reduce the rotation of the pole.

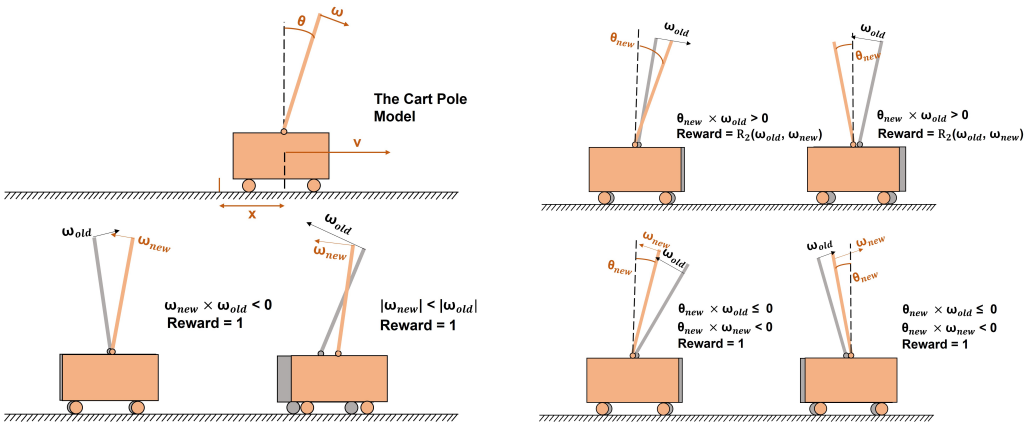
Reward Function 3: This reward function is defined as

$$R_3(\omega_{old}, \omega_{new}, \theta_{old}, \theta_{new}) = \begin{cases} R_2(\omega_{old}, \omega_{new}), & \theta_{new} * \omega_{old} > 0 \\ 1, & \theta_{new} * \omega_{old} \leq 0 \textit{ and } \theta_{new} * \omega_{new} < 0 \\ -1, & \textit{otherwise.} \end{cases} \tag{27}$$

Compared to Reward Function 2, Reward Function 3 considers not only the angular velocity but also the angular position of the pole. In Reward Function 3, Reward Function 2 is only used when ω_{old} and θ_{new} are of the same sign. This is based on the following reasoning. If ω_{old} and θ_{new} are not of the same sign but of opposite signs, θ_{old} must be of the same sign as θ_{new} because

$$\theta_{old} = \theta_{new} - \omega_{old} * \Delta t. \tag{28}$$

Then, θ_{old} and ω_{old} must be of opposite signs. This means that ω_{old} causes a reduction in the angle of tilt of the pole. Therefore, Reward Function 2, which rewards the reversal or reduction of ω_{old}, is no longer suitable. In this case, a comparison of the sign of θ_{new} with that of ω_{new} remains to be further considered. If the signs are opposite, the new state has retained the desired direction of ω_{new} to correct the tilting angle θ_{new}. A positive value then rewards the action taken. Otherwise, the action is punished with a negative value of -1. This reward function is illustrated by Figure 4b.



(**a**) (**b**)

Figure 4. Different reward function designs: (**a**) Reward Function 2 employs the angular velocity of the pole; (**b**) Reward Function 3 employs the angular velocity of the pole as well as the angle of the pole.

4.4. Exploitation and Exploration

With the above R-STDP learning rules and the cart-pole state encoding scheme, the SNN will only be able to learn to choose a good action for states that the agent has already visited. It would not provide a good policy for the states it has not seen yet. To encourage the exploration of various possible states during the initial training phase, we define a probability $P_{explore}$ for the agent to randomly take one of the two possible actions (exploration) instead of taking action determined from the SNN's outputs (exploitation). The value of $P_{explore}$ is set to one at the beginning of the learning. It is downscaled by a value of $\gamma_{explore}$ = 0.9 at the beginning of each new epoch so that as the training process progresses, fewer random actions will be taken, and the learning process can eventually converge.

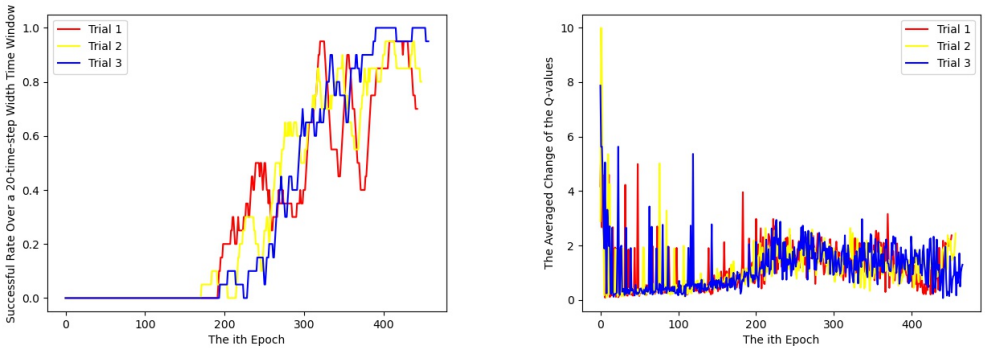
5. The SNN Simulator

This work uses the highly flexible and easily extensible SNN simulator *Brian*2 developed by the computational neuroscience community. Brian2 is a clock-driven simulator (also supports event-driven simulation mode) implemented purely in Python and supports a range of different platforms [55]. It provides a variety of libraries to generate spike signals, create neuron groups, define connecting synapses, monitor spike signals, and control overall simulations. In this study, the input neurons have been defined using the Brian2 function $SpikeGeneratorGroup()$, and the output neuron groups have been defined using the Brian2 function $NeuronGroup()$. Synapses have been created using the Brian2 function $Synapses()$ and connected using the function $Synapse.connect()$. For the TD-STDP SNN, n_{output}=10. For the SD-STDP SNN, n_{output}=1. The counting of spikes generated by output neuron groups has been monitored with the Brian2 function $SpikeMonitor()$.

6. Results

6.1. TD-STDP Learning

During one training session, the duration starting from the initializing/resetting of the gym cart-pole environment to the termination of the cart-pole environment is considered as one epoch. As the goal of the cart-pole problem is to keep the pole upright as long as possible, we define a trial of the cart-pole balancing problem to be successful if the number of time steps by which the pole does not fall is more significant than a threshold of $n_{Threshold} = 200$ time steps. At a particular training stage, we estimate the performance of the SNN at that training stage by considering the success rate over 20 consecutive epochs centered around that stage. Figure 5a shows the success rates as functions of the number of training epochs in three independent runs.



(**a**) (**b**)

Figure 5. The training results of the SNN using TD-STDP, where the three differently colored lines indicate the three independent trials: (**a**) the success rate of the SNN controller changes over the training epochs using TD-STDP learning; (**b**) the root mean square change of the SNN-produced Q values between two consecutive training epochs during TD-STDP training. In this and the subsequent figures, different colors indicate independent training runs.

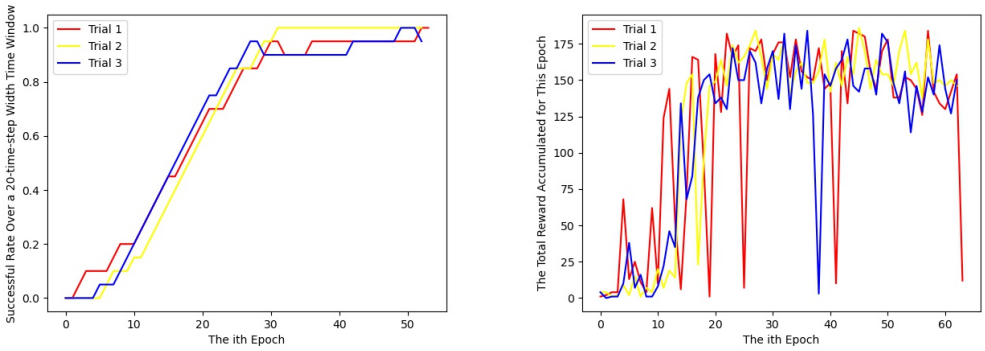
As the training process evolves, the success rate of the SNN controller generally increases, but with several drops. These drops could result from the agent encountering states for which it does not yet have enough experience to form a good policy. This change in success rate indicates that the training process is indeed converging, although somewhat noisy. This phenomenon can also be seen in Figure 5b, which plots the root mean square change of the Q values between two consecutive training epochs against the number of training epochs. The curves exhibit fluctuations within a specific range even after prolonged training.

6.2. R-STDP Learning

We tested the three different reward functions described above. We only show the results for Reward Function 3, in which the rewards depend jointly on angular position and velocity changes. This function led to much higher success rates after training than the other two reward functions.

Using the success rate over 20 epochs as the metric for the performance of the SNN at different training stages, the results of R-STDP are plotted in Figure 6a. Figure 6a, compared with Figure 5a, shows a more satisfactory result for the use of R-STDP. As the training progresses, the success rate increases relatively smoothly. They gradually reached the value of 1 with only tiny fluctuations in less than 50 epochs. In comparison, the TD-STDP training did not show a trend to convergence until more than 300 epochs and remained imperfect and noisy until more than 400 epochs.

The accumulated reward in each training epoch is shown in Figure 6b. As intended, the R-STDP learning process led to a gradual increase in the total rewards.



(**a**) (**b**)

Figure 6. The training results of the SNN using R-STDP and Reward Function 3: (**a**) the success rate of the SNN controller changes over the training epochs using R-STDP learning; (**b**) the total reward accumulated in one epoch versus the number of training epochs.

7. Discussion

TD-STDP is built on the theoretical basis of classical reinforcement learning algorithms. Although for the problem examined here, the TD-STDP approach exhibits lower learning efficiency than R-STDP (R-STDP produces consistently good results after less than 50 epochs, while TD-STDP still delivers noisy results after 300 training epochs), TD-STDP does not use any assumption about the dynamics of the cart-pole system for learning. Thus, it provides a more general solution for RL problems using SNN. The shared basic concepts between traditional RL and TD-STDP-based SNNs also imply that future optimization of SNN for RL may benefit from ideas developed in previous studies on RL.

The interaction between the agent and the environment in which it roams is often crucial in RL control applications. Due to the nature of this application, the reward received by the agent for taking action is usually delayed relative to the timing of the action because the feedback of the environment is obtainable, as the sensory data are usually reactions to previously taken actions. The R-STDP learning rule employed in the cart-pole controller training successfully used the delayed reward signals to extract useful information to help the SNN generate a stable policy. This was achieved by storing the eligibility traces of the synapses and using a reward function jointly determined by current and past observations.

The solutions we presented here for the cart-pole example employ a rather simple but effective SNN structure at the expense of using discrete variables to represent the states of the cart pole. The network has only two layers without using any hidden layers. However, the goal of balancing the pole is still efficiently achieved after a reasonable amount of training. That said, we note that the success of the R-STDP SNN critically relied on the definition of a reward function (here, Reward Function 3) that utilized substantial preestablished knowledge or understanding of the dynamics of the problem. This restricts the model's applicability to general control problems for which learning has to be based on experiences, not on assumptions about the actual dynamics of the system to be the controller. In this sense, TD-STDP presents a more general framework for the SNN to solve control problems, as no pre-established knowledge about the dynamics of the problem is needed. However, for particular problems, such generality may come with the expenses of lower performance relative to R-STDP learning with problem-specific rewards.

The one-hot coding of the state space also proves its usefulness in this application. However, the resolution for discretizing the state space may affect the control system's performance. Moreover, as the dimension of the state space increases, the required number of discretized states (and thus the number of input neurons to cover the state space) will increase exponentially. Implementing other coding schemes for input states will be an interesting problem for future investigations.

This work has only provided a glimpse into the feasibility of using SNN to solve RL problems. The cart-pole game acts as a good starting point for more complicated control problems that could be solved using RL techniques, where SNN has shown its interesting properties that the formation of the synapse connection is heavily affected by the reward function and coding scheme design. These properties could be better explained by a more in-depth study of the observations of neuroscience experiments.

Although RL has been a well-studied field, as a wide range of RL algorithm-based applications have shown their power in various fields, the training and application of SNN in solving RL problems still remain an active field for ML researchers to explore, since these two techniques share the methodology inspired from the learning process of a biological neural system. The long-term goal of this project is to help develop control software in neuromorphic computational units. The current implementation uses an SNN software simulator. Thus, the computational cost of the current implementation cannot reflect the gain in energy efficiency of the SNN. An important follow-up work would be to implement similar SNNs on physical hardware beyond software simulators. The potential of implementing SNN at the hardware level to exploit its power efficiency and biological precision of SNN compared to former ANNs also makes SNN a promising approach to further enhance the overall performance of neural network models.

Author Contributions: Conceptualization, W.P. and Y.L.; methodology, software, writing—original draft preparation, Y.L.; writing—review and editing, supervision, W.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The authors confirm that the data supporting the findings of this study are available within the article

Acknowledgments: We thank Haiyan Liu (University of Science and Technology of China) for his constructive suggestions on the work of this paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Moe, S.; Rustad, A.M.; Hanssen, K.G. Machine Learning in Control Systems: An Overview of the State of the Art. In Proceedings of the Artificial Intelligence XXXV, Cambridge, UK, 11–13 December 2018; Bramer, M.; Petridis, M., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 250–265.
2. Brosilow, C.; Joseph, B. *Techniques of Model-Based Control*; Prentice Hall Professional: Hoboken, NJ, USA, 2002.
3. Fliess, M.; Join, C. Model-Free Control. *Int. J. Control* **2013**, *86*, 2228–2252.
4. Jordan, M.I.; Mitchell, T.M. Machine Learning: Trends, Perspectives, and Prospects. *Science* **2015**, *349*, 255–260, https://doi.org/10.1126/science.aaa8415.
5. Mokhtari, S.; Abbaspour, A.; Yen, K.K.; Sargolzaei, A. A Machine Learning Approach for Anomaly Detection in Industrial Control Systems Based on Measurement Data. *Electronics* **2021**, *10*, 407. https://doi.org/10.3390/electronics10040407.
6. Parekh, D.; Poddar, N.; Rajpurkar, A.; Chahal, M.; Kumar, N.; Joshi, G.P.; Cho, W. A Review on Autonomous Vehicles: Progress, Methods and Challenges. *Electronics* **2022**, *11*, 2162. https://doi.org/10.3390/electronics11142162.
7. Wang, S.; Chaovalitwongse, W.; Babuska, R. Machine Learning Algorithms in Bipedal Robot Control. *IEEE Trans. Syst. Man, Cybern. Part C (Appl. Rev.)* **2012**, *42*, 728–743. https://doi.org/10.1109/TSMCC.2012.2186565.
8. Giusti, A.; Guzzi, J.; Cireşan, D.C.; He, F.L.; Rodríguez, J.P.; Fontana, F.; Faessler, M.; Forster, C.; Schmidhuber, J.; Caro, G.D.; et al. A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots. *IEEE Robot. Autom. Lett.* **2016**, *1*, 661–667. https://doi.org/10.1109/LRA.2015.2509024.
9. Rosenblatt, F. The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain. *Psychol. Rev.* **1958**, *65*, 386.
10. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444.
11. Wu, Z.; Tran, A.; Rincon, D.; Christofides, P.D. Machine Learning-Based Predictive Control of Nonlinear Processes. Part I: Theory. *AIChE J.* **2019**, *65*, e16729.
12. Kaelbling, L.P.; Littman, M.L.; Moore, A.W. Reinforcement Learning: A Survey. *J. Artif. Intell. Res.* **1996**, *4*, 237–285.
13. Barto, A.G. Reinforcement Learning Control. *Curr. Opin. Neurobiol.* **1994**, *4*, 888–893.
14. Zou, J.; Han, Y.; So, S.S. Overview of Artificial Neural Networks. *Artif. Neural Netw.* **2008**, *458*, 14–22.

15. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing Atari with Deep Reinforcement Learning. *arXiv* **2013**, arXiv:1312.5602.
16. Hassabis, D.; Kumaran, D.; Summerfield, C.; Botvinick, M. Neuroscience-inspired Artificial Intelligence. *Neuron* **2017**, *95*, 245–258.
17. Rieke, F.; Warland, D.; Van Steveninck, R.d.R.; Bialek, W. *Spikes: Exploring the Neural Code*; MIT Press: Cambridge, MA, USA, 1999.
18. Levitan, I.B.; Levitan, I.B.; Kaczmarek, L.K. *The Neuron: Cell and Molecular Biology*; Oxford University Press: New York, NY, USA, 2002.
19. Eshraghian, J.K.; Ward, M.; Neftci, E.; Wang, X.; Lenz, G.; Dwivedi, G.; Bennamoun, M.; Jeong, D.S.; Lu, W.D. Training Spiking Neural Networks Using Lessons From Deep Learning. *arXiv* **2021**, arXiv:2109.12894.
20. Vreeken, J. *Spiking Neural Networks, An Introduction*; Adaptive Intelligence Laboratory, Intelligent Systems Group, Utrecht University: Utrecht, The Netherlands, 2003.
21. Kim, S.; Park, S.; Na, B.; Yoon, S. Spiking-Yolo: Spiking Neural Network for Energy-Efficient Object Detection. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020; Volume 34, pp. 11270–11277.
22. Kabilan, R.; Muthukumaran, N. A Neuromorphic Model for Image Recognition using SNN. In Proceedings of the 2021 6th International Conference on Inventive Computation Technologies (ICICT), Coimbatore, India, 20–22 January 2021; pp. 720–725.
23. Mead, C. Neuromorphic Electronic Systems. *Proc. IEEE* **1990**, *78*, 1629–1636.
24. Khan, M.M.; Lester, D.R.; Plana, L.A.; Rast, A.; Jin, X.; Painkras, E.; Furber, S.B. SpiNNaker: Mapping Neural Networks onto a Massively-Parallel Chip Multiprocessor. In Proceedings of the 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), Hong Kong, China, 1–6 June 2008; pp. 2849–2856.
25. Bellman, R. Dynamic Programming. *Science* **1966**, *153*, 34–37. https://doi.org/10.1126/science.153.3731.34.
26. Watkins, C.J.; Dayan, P. Q-Learning. *Mach. Learn.* **1992**, *8*, 279–292.
27. Gerstner, W.; Kistler, W.M.; Naud, R.; Paninski, L. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*; Cambridge University Press: Cambridge, UK, 2014.
28. Izhikevich, E.M. Which Model to Use for Cortical Spiking Neurons? *IEEE Trans. Neural Netw.* **2004**, *15*, 1063–1070.
29. Brunel, N.; Van Rossum, M.C. Quantitative Investigations of Electrical Nerve Excitation Treated as Polarization. *Biol. Cybern.* **2007**, *97*, 341–349.
30. Ghosh-Dastidar, S.; Adeli, H. Spiking Neural Networks. *Int. J. Neural Syst.* **2009**, *19*, 295–308.
31. Lecun, Y. A Theoretical Framework for Back-Propagation. In Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA, USA, 17–26 June 1988; Touretzky, D., Hinton, G., Sejnowski, T., Eds.; Morgan Kaufmann: Burlington, MA, USA, 1988; pp. 21–28.
32. Ledinauskas, E.; Ruseckas, J.; Juršėnas, A.; Buračas, G. Training Deep Spiking Neural Networks. *arXiv* **2020**, arXiv:2006.04436.
33. Ding, J.; Yu, Z.; Tian, Y.; Huang, T. Optimal ANN-SNN Conversion for Fast and Accurate Inference in Deep Spiking Neural Networks. *arXiv* **2021**, arXiv:2105.11654.
34. Rueckauer, B.; Liu, S.C. Conversion of Analog to Spiking Neural Networks Using Sparse Temporal Coding. In Proceedings of the 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 27–30 May 2018; pp. 1–5.
35. Wu, Y.; Deng, L.; Li, G.; Zhu, J.; Shi, L. Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks. *Front. Neurosci.* **2018**, *12*, 331.
36. Bohte, S.M.; Kok, J.N.; La Poutré, J.A. SpikeProp: Backpropagation for Networks of Spiking Neurons. In Proceedings of the ESANN, Bruges, Belgium, 26–28 April 2000; Volume 48, pp. 419–424.
37. Markram, H.; Gerstner, W.; Sjöström, P.J. A History of Spike-Timing-Dependent Plasticity. *Front. Synaptic Neurosci.* **2011**, *3*, 4.
38. Hebb, D.O.; Penfield, W. Human Behavior After Extensive Bilateral Removal from the Frontal Lobes. *Arch. Neurol. Psychiatry* **1940**, *44*, 421–438.
39. Song, S.; Miller, K.D.; Abbott, L.F. Competitive Hebbian Learning Through Spike-Timing-Dependent Synaptic Plasticity. *Nat. Neurosci.* **2000**, *3*, 919–926.
40. Diehl, P.U.; Cook, M. Unsupervised Learning of Digit Recognition Using Spike-Timing-Dependent Plasticity. *Front. Comput. Neurosci.* **2015**, *9*, 99.
41. Bi, G.Q.; Poo, M.M. Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type. *J. Neurosci.* **1998**, *18*, 10464–10472.
42. Scellier, B.; Bengio, Y. Equilibrium Propagation: Bridging the Gap between Energy-based Models and Backpropagation. *Front. Comput. Neurosci.* **2017**, *11*, 24.
43. Frémaux, N.; Gerstner, W. Neuromodulated Spike-Timing-Dependent Plasticity, and Theory of Three-Factor Learning Rules. *Front. Neural Circuits* **2016**, *9*, 85.
44. Florian, R.V. Reinforcement Learning Through Modulation of Spike-Timing-Dependent Synaptic Plasticity. *Neural Comput.* **2007**, *19*, 1468–1502.
45. Schultz, W.; Dayan, P.; Montague, P.R. A Neural Substrate of Prediction and Reward. *Science* **1997**, *275*, 1593–1599.
46. Bargmann, C.I. Beyond the Connectome: How Neuromodulators Shape Neural Circuits. *Bioessays* **2012**, *34*, 458–465.
47. Frémaux, N.; Sprekeler, H.; Gerstner, W. Reinforcement Learning Using a Continuous Time Actor-Critic Framework with Spiking Neurons. *PLoS Comput. Biol.* **2013**, *9*, e1003024.
48. Geva, S.; Sitte, J. A Cartpole Experiment Benchmark for Trainable Controllers. *IEEE Control. Syst. Mag.* **1993**, *13*, 40–51.

49. Rafe, A.W.; Garcia, J.A.; Raffe, W.L. Exploration Of Encoding And Decoding Methods For Spiking Neural Networks On The Cart Pole And Lunar Lander Problems Using Evolutionary Training. In Proceedings of the 2021 IEEE Congress on Evolutionary Computation (CEC), Kraków, Poland, 28 June–1 July 2021; pp. 498–505.
50. Ding, Y.; Zhang, Y.; Zhang, X.; Chen, P.; Zhang, Z.; Yang, Y.; Cheng, L.; Mu, C.; Wang, M.; Xiang, D.; et al. Engineering Spiking Neurons Using Threshold Switching Devices for High-efficient Neuromorphic Computing. *Front. Neurosci.* **2022**, *15*, 1732.
51. Zhou, Y.; Wang, Y.; Zhuge, F.; Guo, J.; Ma, S.; Wang, J.; Tang, Z.; Li, Y.; Miao, X.; He, Y.; et al. A Reconfigurable Two-WSe2-Transistor Synaptic Cell for Reinforcement Learning. *Adv. Mater.* **2022**, *34*, 2107754.
52. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. OpenAI Gym. *arXiv* **2016**, arXiv:1606.01540.
53. Seger, C. An Investigation of Categorical Variable Encoding Techniques in Machine Learning: Binary Versus One-Hot and Feature Hashing. Dissertation, KTH, Stockholm, Sweden, 2018.
54. Song, S.; Abbott, L.F. Cortical Development and Remapping Through Spike Timing-Dependent Plasticity. *Neuron* **2001**, *32*, 339–350.
55. Stimberg, M.; Brette, R.; Goodman, D.F. Brian 2, An Intuitive and Efficient Neural Simulator. *Elife* **2019**, *8*, e47314.