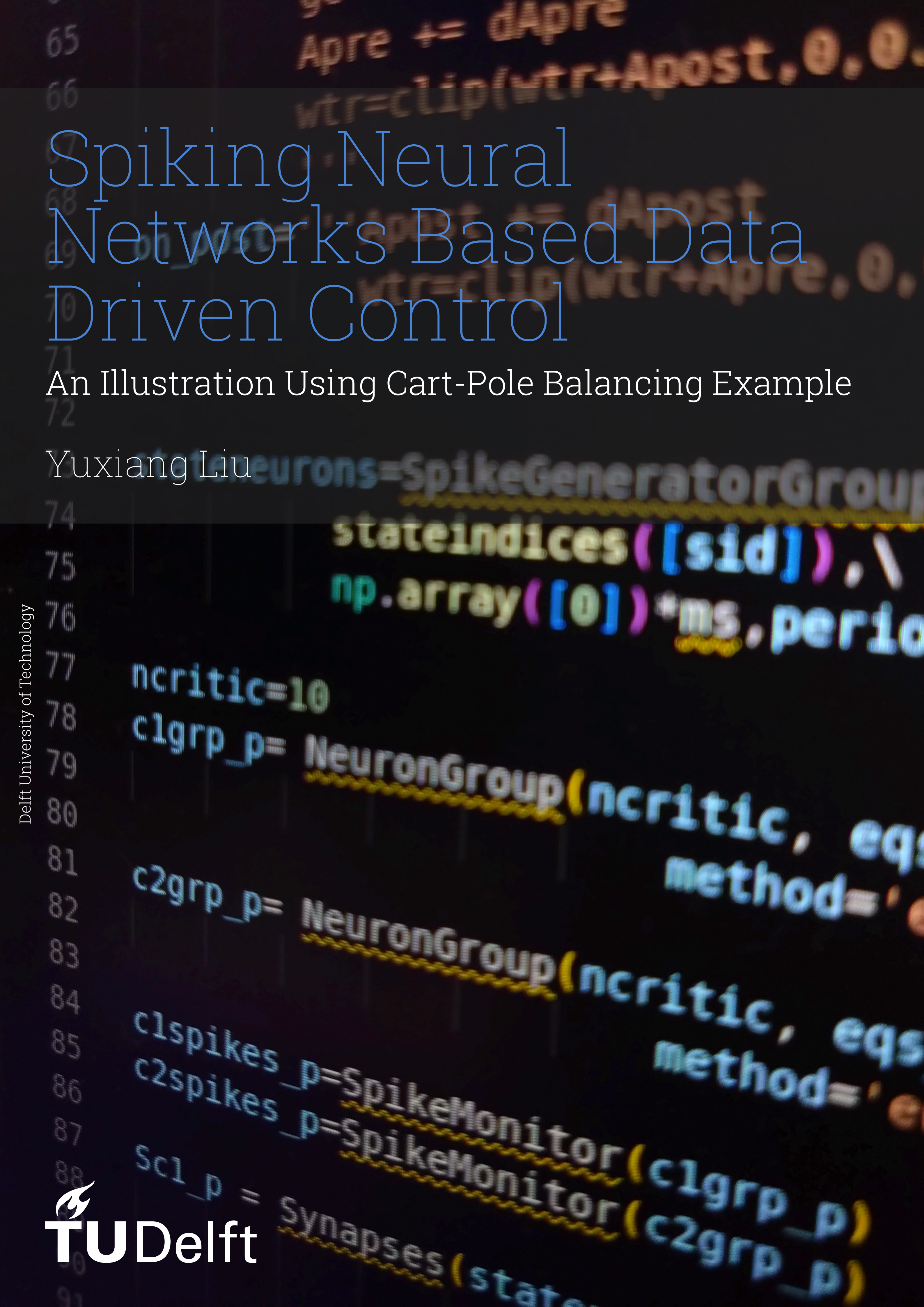


Spiking Neural Networks Based Data Driven Control

An Illustration Using Cart-Pole Balancing Example

Yuxiang Liu

Delft University of Technology

TU Delft

Spiking Neural Networks Based Data Driven Control

An Illustration Using Cart-Pole Balancing Example

by

Yuxiang Liu

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on June 19th, 2023 at 10 AM.

Student number:	5055547	
Project duration:	March 1, 2022 – June 12, 2023	
Thesis committee:	Prof. Martijn Wisse,	CoR, 3ME
	Dr. Rajendra Bishnoi,	Q&Ce, EEMCS
	Dr. Wei Pan,	CoR, 3ME (recently moved to the University of Manchester)

TUDelft

Summary

Machine learning can be effectively applied in control loops to robustly make optimal control decisions. 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 hardwares are being rapidly developed. A defining character of control problems is that environmental reactions and delayed rewards must be considered. While reinforcement learning (RL) provides the fundamental mechanisms to address such problems, realizing these mechanisms in SNN learning has been underexplored. Previously, schemes of spike timing dependent plasticity (STDP) learning 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. While the TD-based learning rules are very general ones, the resulted 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 learnt much more efficiently than by TD-STDP. The work of this master thesis project has also been published as a paper in *Electronics, Vol. 12* [45].

Abbreviations

Abbreviation	Definition
ANN	Artificial Neural Network
ML	Machine Learning
RL	Reinforcement Learning
R-STDP	Reward-modulated Spike Timing Dependent Synaptic Plasticity
SNN	Spiking Neural Network
STDP	Spike Timing Dependent Synaptic Plasticity
TD	Temporal Difference
TD-STDP	Temporal Difference-modulated Spike Timing Dependent Synaptic Plasticity

Contents

1 Introduction

There is substantial interest in using Spiking Neural Networks (SNNs), a framework of machine learning (ML) closely related to biological neural systems, for control problems because of the potential high energy efficiency of SNN relative to other ML approaches like Artificial Neural Networks (ANNs). The main topic of this thesis is about the designing and testing of SNN algorithms that implement reinforcement learning (RL), which is one of the most widely applied ML approaches to control tasks.

1.1. Control Tasks and Machine Learning

The past decade has witnessed the vast development of ML, which is making substantial impacts in various disciplines of science and engineering [37]. In the particular field of control engineering, ML approaches have also been extensively investigated for tasks that include industrial process control[50], autonomous vehicle maneuvering [52], and robotics operation [67].

As summarized by Moe et al. [49], a traditional control loop can be considered as comprising several essential blocks, including the internal and external observation blocks, the control blocks, and the (optional) dynamics 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 blocks to determine what 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 [12], the decision making of the control blocks is assisted by the dynamics model block, which predicts the behavior of the system (e.g., through simulations of the physical system), while in model-free control [20], no dynamic model of the system is required.

For many control systems, blocks fulfilling one or more of the above functions can be advantageously implemented with ML. For example, ML can be applied to preprocess complex sensory data before these data are used to make control decisions [29]. Moreover, supervised learning (one type of ML) [35] can be applied to produce a dynamic model [71], 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 core of the control loop, ML can be used to achieve robust nonlinear mapping from the high-dimensional state variables to optimal control decisions. In fact, reinforcement learning (RL) [39], a subfield of machine learning, has been widely recognized as targeting problems that generally fall into the scope of control engineering [5].

1.2. Machine Learning with Artificial Neural Networks

So far, machine learning approaches in control engineering have been mainly based on artificial neural networks (ANNs) [34], because ANN models are of high adaptability, high flexibility, and easy to train. A typical ANN consists of multiple layers of artificial ”neurons” or nodes [73]. The nodes in the first layer (i.e., the input layer) receive data from outside of the ANN and pass these data to the internal nodes. Nodes in subsequent layers receive output from nodes of earlier layers through predefined internode connections. Each node combines multiple inputs using (learnable) connection-specific weights, passes the result (possibly after adding a learnable bias) through a nonlinear activation function, and produces output. The outputs of the nodes in the last layer usually correspond to the final output of the ANN.

To train an ANN, items of training data are propagated through the network layers. Then the task-specific loss functions are evaluated based on the output of the network, and the derivatives(or gradient) of the loss with respect to the learnable network parameters are computed with the back-propagation algorithm [42]. The parameters are updated according to the gradient to reduce the loss (i.e., gradient descend or GD [55]).

1.3. Machine Learning with Networks of Spiking Neurons

The wide applicability of deep ANNs (i.e., ANNs with many hidden layers) to different problems can be attributed, at least partially, to their hierarchical organization of relatively simple computational units into dedicated network structures. Such structures have been inspired by the intricately connected neuron networks in a biological brain. However, the computations performed by the nodes of ANNs (the artificial ”neurons”) are fundamentally different from the ”computations” performed by the biological neurons. In fact, Vreeken has proposed that computational neuron networks can be divided into three generations [65]: the first and second generations use artificial neurons that accept real-valued inputs and generate outputs of binary (the first generation) or continuous (the 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 true neurons in biological brains [54].

A typical neuron cell has many dendrites and an axon, which are respectively connected to upstream and downstream neurons through micro-structures called synapses [44]. The interconnected neurons use spikes to communicate with each other [54]. Over a period of time, a series of spikes (a spike train) generated by a pre-synaptic neuron can modulate the voltage level of a post-synaptic neuron. When the latter voltage is above a threshold value, the post-synaptic neuron generates its own spike (i.e. firing) and its voltage level drops down. In a spiking neuron network (SNN), each neuron will receive the spike trains from the upstream neurons connecting to its input synapses, produces its own spike train, and passes its spike train to the downstream neurons connected with its output synapses.

It has been suggested that SNNs can offer much higher energy efficiency than conventional ANNs[40]. Neuronal 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 that needs to be always active. Energy-efficient SNN computations are actively being pursued through the development of both SNN-based ML algorithms [38] and neuromorphic hardwares that enable on-chip SNNs [48].

A major current difficulty in using SNN is the lack of general learning algorithms for various ML tasks [28]. Because the spike train signals in SNNs are discrete and nondifferentiable, backpropagation training cannot be straightforwardly applied to SNN. It is an active field of research to develop appropriate algorithms to train SNNs for different applications. These algorithms have been roughly classified as conversion-based and spike-based [18]. In a conversion-based algorithm, a trained ANN is converted to an SNN so that the SNN can generate the same input-output mapping as the ANN. It has been shown that this conversion approach can produce SNNs of comparable accuracy as deep ANNs for image recognition tasks. In another type of spike-based algorithms, gradient descend training of SNN is enabled using differentiable approximation of the spike function for derivative estimation and backpropagation. This approach has produced SNNs that can solve small-scale image recognition problems.

Another highly interesting type of SNN learning algorithms make use of the so-called spike timing dependent plasticity (STDP) of synapses [46]. Compared with the other algorithms, this type of algorithms more resemble the adaptive learning process of a biological brain, and are potentially more suitable for energy-efficient implementation on a neuromorphic device. However, the development of STDP rules for SNN learning remains an underexplored research direction, especially in robotics and autonomous control [1, 8], where adaptive learning and energy efficiency are often emphasized.

1.4. Focuses and Structure of the Current Report

In the current thesis, we investigate the use of SNN instead of ANN in control problems. In particular, we are interested in designing SNNs to perform the task of making control decisions that maximize the future rewards associated with the decisions. While the most widely used ML approach for such tasks is reinforcement learning (RL), the most natural learning method for SNN is perhaps spike timing dependent plasticity (STDP). For reinforcement learning, the STDP-based approaches include introducing either reward signals or temporal difference signals into the algorithm (i.e. R-STDP or TD-STDP). Both algorithms have been proposed in previous studies. However, the original work introducing R-STDP has only used regression tasks without delayed rewards as demonstrating examples. For TD-STDP, while actor-critic reinforcement learning models have been implemented in previous works, there lack reports of realization of other RL models, including Q-learning, which can be considered as one of the most fundamental RL techniques. Moreover, the relative advantages and drawbacks of these two variants of STDP have not been analyzed by considering the same problem.

Here we try to address the above issues by considering an example of using SNN to solve the classical Cart-Pole Game. With a simple yet effective SNN architecture, we realized Q-learning for the Cart-Pole control task with TD-STDP. We also realized efficient R-STDP through introducing delayed rewards. We examined the effects of learning rates and exploration/exploitation schemes in our TD-STDP algorithm. We also demonstrated the importance of a proper reward function in R-STDP. We found that while R-STDP could be less general than TD-STDP because to design a good reward function for R-STDP requires prior understanding of how the environment respond to control decisions, learning with R-STDP can be much more efficient if a good reward function can indeed be defined.

The rest of this thesis is structured as below. In Chapter 2, we present brief introductions of the basic techniques of RL, followed by an overview of SNN with emphasis on TD-STDP and R-STDP learning. The Cart-pole game is introduced at the end of Chapter 2. Chapters 3 and 4 present respectively methods and results of our work in details. Chapter 5 contains concluding remarks.

2 Preliminary

In this chapter, we give preliminary introductions to several major components of our work. They are reinforcement learning (RL), spiking neural network (SNN) and the Cart-Pole Game.

2.1. A Brief Introduction to Reinforcement Learning (RL)

The idea of reinforcement learning (RL) arises from the study of classical and instrumental conditioning of animals. It emphasizes the interaction between an learner and the environment surrounding the learner as well as the use of a feedback system that evaluates the learner's performance. The feedback will then be used by the learner to adjust its behavior to achieve certain preset objectives.

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 action that the agent takes can potentially change the state that the agent currently stays in. At each state, the agent may receive a certain amount of reward. The goal of the agent is to choose a proper action at each given state, so that the total amount of reward the agent received from the states it goes through will eventually be maximized.

The rules by which the agent chooses its action can be noted as the **policy** π. For example, π can correspond to the probability for the agent to take action a when at state s, namely, $\pi(s, a) = P(a|s)$ 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 is able to receive by following the policy π at all future states, namely,

$$Q_\pi(s_n, a_n) = E_\pi(G|s = s_n, a = a_n), \tag{2.1}$$

in which G stands for the total future rewards or gaining, E_π stands for averaging over all possible evolution 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 the action with the maximum future gaining after step $n + 1$, Q_{opt} should satisfy the following *Bellman Optimal Equations*, which were proposed by Bellman as the basis for solving the RL problem via a dynamic programming [6] algorithm,

$$Q_{opt}(s, a) = R(s) + \Sigma_{s' \in S} P^a_{ss'} max_{a' \in A} Q_{opt}(s', a'), \tag{2.2}$$

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 from some initial Q-value function and iteratively update it. In this process, we can define the deviations of the current Q-value function from the Bellman equations as the temporal differences, namely,

$$TD_Q = R(s) + \Sigma_{s' \in S} P^a_{ss'} max_{a' \in A} Q(s', a') - Q(s, a) \tag{2.3}$$

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

$$Q_{new}(s,a) = Q_{old}(s,a) + \gamma TD_Q, \tag{2.4}$$

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** [69]. When the state space and the action space are discrete, the Q-value function is actually 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 to take based on the states in which it is in. It has been proven by its inventor Watkins that the Q-learning algorithm based on equation 2.4 can converge if the Q-values are represented in the form of a lookup table [68].

2.2. An Overview of SNN

2.2.1. Spiking Neuron Models

SNN is built using the third generation of artificial neurons (i.e., the spiking neuron), which are capable of processing individual spikes. Various spiking neuron models have been proposed over the years and the differences of these models showed a trade-off between the accuracy of the described biological process and computational efficiency [33]. The Leaky Integrate and Fire (LIF) Model is one of the widely used spiking neuron models that are computationally inexpensive while still being able to capture the most essential characteristics of a biological neuron as a unit to process the temporal information.

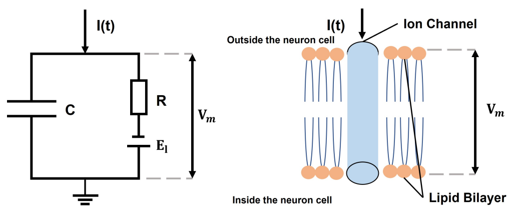


Figure 2.1: The change of the membrane potential of a neuron cell can be modeled as the response of the RC electronic circuit shown in the figure above.

The LIF Model was proposed by Louis Lapicuqe in the year of 1907 when he discovered that the voltage level (i.e., the membrane potential) change inside a biological neuron cell can be modeled by the voltage level change inside a RC low pass filter shown in Figure 2.1 [13]. The dynamics of the LIF model can be formulated as Equation 2.5,

$$C\frac{dv_m}{dt} = -g(v_m - E_l) + i(t), \tag{2.5}$$

where v_m is the membrane potential of the neruon cell, C and g is respectively the capacitance and the conductance of the abstracted RC low pass filter circuit, E_l is the resting potential of the neuron cell,and $i(t)$ stands for the time-varying current that flows into the neuron.

This equation can be derived by using simple Ohm's law as the differential form of a capacitor equation. Briefly, the current flows through the resistor can be calculated as

$$i_r = g(v_m - E_l). \tag{2.6}$$

The current flows through the capacitor is

$$i_c = C\frac{dv_m}{dt} \tag{2.7}$$

The sum of i_r and i_c should be equal to the total input current $i(t)$, which leads to Equation 2.5.

The LIF model mainly focuses on describing the time integral of current that flows into the neuron cell, which builds up the membrane potential of the cell until it reaches over the threshold voltage, which triggers the neuron cell to fire a spike. It also contains a "leaky" term that leads to the restoration of the membrane potential to its resting value, which is caused by the diffusion of ions in a neuron cell [26].

2.2.2. Encoding Information with Spikes

The information transmitted within SNNs is encoded by short electrical pulses (i.e., spikes). Two main categories of spike coding schemes could be found from biological studies: one is rate coding and the other is temporal coding [14]. During the early days of neuroscience, rate coding has been considered as the main scheme used by biological neuron systems to encode sensory information [24]. However, later studies have shown that rate coding would fail to explain certain observations such as the fast response time of a human visual system [63]. The timing of the occurrence of a spike (i.e., temporal coding) was then proposed as the coding scheme for the information processed in the neural system, and this idea has been supported by a wide range of studies over different sensory systems [2]. In some studies population coding has been considered as a third category spike coding scheme. However, population coding is essentially to consider a group of neurons, each of which still uses either rate coding or temporal coding [3].

Rate Coding

Rate coding translates the information to be encoded into the spike firing rates of the spiking neurons. Three types of rate coding could be commonly found in the literature, including count, density, and population rate coding [3].

Count rate is defined by

$$v = \frac{N_{spikes}}{T}, \tag{2.8}$$

where N_{spikes} stands for the spike count and T stands for the time window size. This coding scheme is actually calculating the mean firing rate (i.e., the average number of fired spikes in terms of a period of time), and therefore is sometimes also referred to as frequency coding.

Density rate is defined by

$$p(t) = \frac{1}{\Delta t}\frac{N_{spikes}(t; t+\Delta t)}{K}, \tag{2.9}$$

where N_{spikes} is the number of spikes emitted though time period $(t; t+\Delta t)$, K is the total number of runs, and Δt is time step length for each run. Equation 2.9 is used to calculate the average number of spikes fired over K number of runs.

Population rate is defined by

$$A(t) = \frac{1}{\Delta t}\frac{N_{spikes}(t; t+\Delta t)}{N}, \tag{2.10}$$

where N is the number of neurons in a neuron group and N_{spikes} stands for total number of spikes emitted during time period $(t; t+\Delta t)$. This coding approach computes the average number of spikes fired by a group of N neurons over a time period of length Δt.

Although current studies in neuroscience have suggested that rate coding may not correctly represent the predominant information encoding scheme adopted by a biological brain, rate coding is still widely applied in SNNs for certain benefits. Rate coding usually provides better tolerance to errors relative to temporal coding, since the information is represented by not single but multiple spikes. More number of spikes can also mean that there are stronger stimuli for the network during learning [18]. Researchers have already successfully implemented SNNs using rate coding in tasks including object classification and robotic control [15, 66].

Temporal Coding

Temporal coding translates the information to be encoded into the timing of spikes. Various temporal coding schemes have been proposed over the years, some of them modeling observations from the field of neuroscience.

Time to first spike (TTFS) coding use the time difference between the start of the stimulus and the first coding spike to represent information [36]. The timing of the first spike is usually determined by the inverse of the amplitude of the input, a (e.g., $t = \frac{1}{a}$) or some linear relation to a (e.g., $t = 1 - a$). In this way, a small value of a leads to a late fired spike or no fired spike at all, while a large value of a results in an early fired spike. It is shown by Wenzhe et al. that TTFS could be beneficial in achieving high computational performance with a relatively low hardware implementation overhead [30].

Phase coding translate the information to be encoded into the relative time difference between spikes with an oscillation as a reference [61]. The coded signals are sent out in cycles. Several neurons are used in this coding approach and each of them fires spikes with a periodic pattern if the input is zero (relative to some reference value). The spike timing difference between the oscillating reference and each neuron encodes the input in a way that is similar to TTFS.

Rank-order coding (ROC) encodes the information with the firing order of a group of neurons (with a global timing reference). Unlike TTFS, the precise timing of the spikes is omitted in this coding scheme [64]. For example, a group of spiking neurons (noted respectively as A, B, C, D, and E) can encode an input stimulus as $A > C > E > B > D$ ("$A > C$" means neuron A fired earlier than neuron C) and there are 5! different possible firing orders, each of which can be used to encode a particular input state.

Inter-spike interval (ISI) coding or latency coding encodes the data into the latency between two consecutive spikes [51]. One special case of ISI is **Burst coding**, which uses a group of spikes with a very small ISI (i.e., a burst) and translate the input information into various interspike latency.

Temporal Contrast (TC) coding generates a spike train when change in the input signal is observed. Three commonly applied TC coding algorithms are **threshold-based representation** (TBR), **step-forward** (SF), and **moving-window** (MW). TBR detects the absolute input signal changes and emits either positive or negative (depends on whether the change corresponds to increasing or decreasing) spike signals if the change exceeds a threshold value. SF compares the signal difference between the current input and a moving baseline, and emits a positive or negative spike when the difference exceeds a threshold value. MW takes the mean value of the signal in a predefined time window and emits spikes based on the change of this mean value [53].

Temporal coding, supported by our current observations of the biological neural system, comes with certain advantages. Due to the sparsity of the emitted spikes in temporal coding, the power consumption of using temporal coding can be significantly lower than that of using rate coding. Temporal coding is also able to provide a fast respond time if the information is encoded by the time of a single spike [25]. Several interesting applications of SNN using temporal coding include clustering of high-dimensional data [72], and an artificial microelectronic nose [7].

2.2.3. Training Methods

Due to the richness of its temporal dynamics, a SNN can be trained in several different approaches [19]. Some training methods are adapted from previous studies on ML[62] and some are inspired by the biological process of how a synapse is intensified or weakened [47]. So far there is no learning approach that can dominate the other approaches in all application scenarios. In a particular scenario, one learning approach could be more effective than other approaches.

Convert Pre-trained ANN

One way of training SNN is first to train an ANN with the desired functionality and then to convert the ANN into a SNN. This is also known as shadow training [19]. The conversion process is done by replacing the ANN neurons with spiking neurons. The rule for replacing the neurons could be matching the activation functions of the ANN nodes either to the firing rates of the spiking neurons [17] or to the timing of the spikes emitted by the spiking neurons [56].

An obvious advantage of applying the conversion method is that previous knowledge of how to train ANNs could be directly applied for the training of SNNs. Therefore, problems associated with the training of SNNs with multiple layers (i.e., deep SNNs) could be solved with methods learned from the studies of deep ANNs. Another advantage of the conversion method is that one can avoid the dead neuron problem in directly training SNNs. The dead-neuron problem is that certain neurons may not fire spikes at all during training, and their states do not affect the total loss value. As a result, the weights of the dead neurons could not be updated by training.

The conversion method has several drawbacks. The training of a complex ANN itself is time-consuming, and the conversion process from an ANN to an SNN also requires a rather long simulation time. The performance of the converted SNN is usually worse than that of the source ANN since the conversion process can lead not to an exact replication but only to an approximation of the original ANN [43].

Backward Propagation

Back propagation (BP) is the most widely used learning algorithm for ANN [42]. The goal of BP is to quantify how the change in the weight parameters affect the value of the loss functions. More formally, BP generates the partial derivatives of the loss with respect to the weights (i.e., the gradient). The BP algorithm consists of two stages. The first is feedforward, in which an input is fed into the network and the output loss is calculated. The second stage is to backpropagate the calculated derivatives of the loss from output to input to calculate the gradient by using the chain rule of derivatives.

For applying BP to a SNN, one major challenge is that spikes are discontinuous and therefore could not be differentiated. The SpikeProp method uses the fact that time is continuous and calculates the derivative of the timing of the spike with respect to weight [9]. However, the use of SpikeProp requires that at least one spike be emitted by each neuron, or the derivative cannot be computed. A dead neuron will freeze the learning process. This drawback is often overcome by applying a force firing regularization mechanism.

The Back Propagation Through Time or BPTT method unrolls the total computation graph of the neural network and applies the back propagation framework to it. There are multiple branches for backpropagation in the unrolled computing graph, which can be chosen on the basis of whether a neuron has fired or not. BPTT will only propagate the gradient backward through the computational path connected by neurons that have fired [70].

Synaptic Plasticity

The above algorithms for learning SNNs have been transferred mainly from ANN studies. They do not correspond to the actual way how the learning process takes place in a biological brain. Neuroscience studies have long revealed synaptic plasticity, namely, the changes of the connection strengths at synapses, as a fundamental mechanism underlying biological learning. Naturally, learning algorithms based on synaptic plasticity for SNN have been extensively examined.

The Hebbian plasticity is a type of synaptic plasticity that generally follows the rule first postulated by Hebb as ”Cells that fire together wire together.” [31]. In other words, it relies on the correlated firing activity of the pre- and post-synaptic neurons to decide whether to change the strength of the connectivity of a synapse. The core idea behind these learning rules is to build up synaptic connections to represent causal relationships.

Spike-timing dependent synaptic plasticity (STDP) is a form of Hebbian learning with precisely defined relationships between spike timing and changes in connection strengths [59]. Biologically, STDP is supported by the experimental finding that the co-activation of the pre- and the post-synaptic neurons can lead to two types of plasticity or synaptic changes: long-term potentiation (LTP) and long-term depression (LTD). LTP refers to the strengthening of the synaptic connection when the pre-synaptic neuron fired within a short period before the firing of the post-synaptic neuron, while LTD refers to the weakening of the connection when the firing events took place in a reversed order. Equation 2.11 presents a simple mathematical model for STDP with LTP and LTD,

$$\Delta w = \begin{cases} A_{+}exp(-\frac{s}{\tau_{+}}), & s > 0 \\ A_{-}exp(\frac{s}{\tau_{-}}), & s <= 0, \end{cases} \tag{2.11}$$

where Δw is the change of the connection strength or synaptic weight, s is the timing difference between the post- and pre-synaptic spikes, 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.

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 synaptic plasticity process of SNN [22, 21]. The modulating factor can be determined from the reactions of the environment 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 the optimal outcomes. Neuroscience studies have shown that in biological brains, the chemical substance dopamine is one of the most important neuromodulators in such feedback or reward-based learning processes. Namely, the dopamine level represents the amount of rewards that guide the learning process by modulating STDP [4].This biological phenomena can be simulated with the extension of the STDP model in formula 2.11 into the following modulated STDP formula,

$$\Delta w = M * w_{eligibility}, \tag{2.12}$$

in which M represents the effects of the neuromodulators, while $w_{eligibility}$ selects synapses for the reward-modulated plasticity according to the conventional STDP rules[22].

Several previous studies have investigated the application of the above idea of modulated STDP-learning rules for various ML problems including the making of control decisions [21, 23]. Florian et al. introduced the reward-modulated STDP or R-STDP algorithm [21], in which the modulating factor came from the deviations of the SNN's actual and expected outputs. In that study, the effectiveness of the algorithm was illustrated with the examples of using SNN to solve the XOR problem and to output pre-defined spike trains. We note that these examples were essentially regression problems, in which the so-called "rewards" were actually derived from the errors of the regression but not from a reaction of an environment upon which a control decision (determined from the outcome of the SNN) had impacted. In other words, the rewards for these problems were instantaneous but not delayed, as in typical control problems.

Frémaux et al. have proposed to use the temporal difference (TD) concept in reinforcement learning (RL) as the modulating factor for STDP [23]. This TD-STDP learning rule is applied to train actor-critic-type reinforcement learning (RL) controllers [39, 5]. 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 analyses in ref [23] illustrated the potential of using SNNs to implement general RL strategies for control tasks.

Despite these previous studies, STDP-based SNN learning for control tasks is still an underexplored research problem. As mentioned above, in the original study that examined R-STDP [21], the examples did not comprise typical control tasks with delayed rewards. Moreover, there lack 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 the use of SNN with reward factor-modulated STDP learning for control tasks by considering the Cart-Pole balancing problem [27] 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 [23]. Being based on the RL Q-learning strategy [69], the SNN presented here has a much simpler overall network architecture than the critic-actor architecture of ref [23]. While 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 turns out to lead 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 designing of the reward function, a more robust SNN-based controller can be learned much more efficiently by R-STDP than by TD-STDP.

2.2.4. SNN Simulators

Software simulators for SNN have been implemented to help the development of new algorithms or system architectures based on neuromorphic computing. These include NEST, Brian2, Brian2GeNN, BindsNET and Nengo, which are all openly accessible [41]. In general, these simulators use mainly two types of simulation strategies: one is synchronous ("clock-driven") and the other one is asynchronous("event-driven") [10].

A clock-driven simulator will update the state of each neurons in the network at the same time for each time step (also known as a "clock tick") of the simulation (i.e., $S(t) \rightarrow S(t+dt)$). It is shown by Hirsch and Smale that if the simulated differential equation is linear, the state update process is also linear, which gives us the idea that the task of updating a state is equivalent to multiplying a matrix to the previous state vector (i.e., $S(t) = AS(t+dt)$ for some matrix A). Thus clock-driven simulators could be relatively easily built upon existing matrix-based scientific computation tools [32]. When the simulated differential equations are nonlinear, numerical integration methods such as the Euler's method or the Runge–Kutta method can be applied. The accuracy of the simulated results highly relies on the granularity of the time step (i.e., how long the "clock tick" would be) since the spike timing is strictly aligned to a fixed gird of "clock ticks". Therefore, the result of a clock-driven simulator is a discrete-time approximation to the actual temperal dynamics even the differential equations can be integrated with very high accuracy [10].

An event-driven simulator will only update the state of a neuron at the time when it detects an input spike or fires an output spike. The inter-spike dynamics is integrated analytically. Thus, an event-driven simulator can provide relatively more accurate simulation results than a clock-driven simulator, but at the expense of the need for more computational resources and is thus usually only possible for small-scale network models [16].

In this work, the highly flexible and easily extensible SNN simulator Brian2 developed by the computational neuroscience community is used. Brian2 is a clock-driven simulator (also supports event-driven simulation mode) implemented purely in Python and supports a range of different platforms [60]. It provides a variety of libraries to generate spike signals, create neuron groups, define connecting synapses, monitor spike signals, and control the overall simulations. In this study, the input neurons have been defined by using the Brian2 function $SpikeGeneratorGroup()$, the output neuron groups have been defined by using the Brian2 function $NeuronGroup()$. The synapses have been created by 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 the output neuron groups has been monitored with the Brian2 function $SpikeMonitor()$.

2.3. The Cart-Pole Game

The Cart-Pole Game is one of the classical control problems that have been widely applied for the performance evaluation of control algorithms. This problem could be stated as that given a cart with a pole attached to it through a fixed joint (i.e., the pole can only rotates 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 goal of the controller is that the pole should be kept 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 the work of this thesis, the Cart-Pole model environment implemented in the gym library [11] is applied. This environment would terminate its simulation if one of these three situations occurs:

- The pole falls down, which occurs when the absolute value of the angle of the pole is larger than $12°$.
- The cart slides out of 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) value.

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

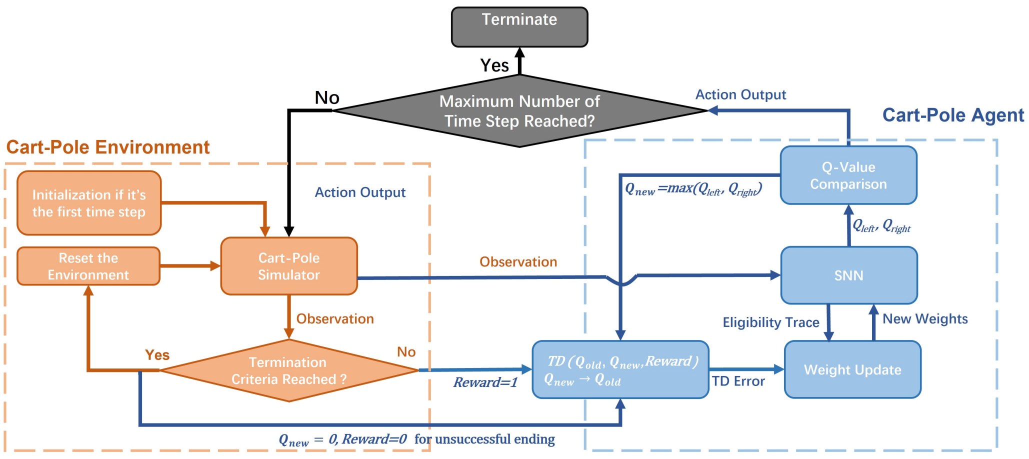
3
Methodology

In this chapter, we present the computational models and methods used in our work in details. We describe the overall workflows of the TD-STDP and R-STDP learning processes, the structure of the SNN including input and output spike encoding, formulations of the learning rules, and so on.

3.1. The TD-STDP SNN

3.1.1. The Overall Workflow of the Program

The overall framework of our program using TD-STDP learning to solve the Cart-Pole control problem is presented in Figure 3.1. The entire program can be divided into two main parts, one is the Cart-Pole environment, the other is the SNN-based controller. Each training episode comprises consecutive a simulation of the Cart-Pole steps from start till one of the finishing conditions is met. At each Cart-Pole step, the environment simulator and the SNN simulator run in sequential order. 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 of time, and produces its output, based on which an action to be applied to the Cart-Pole environment will be chosen. After taking the action, the Cart-Pole environment evolves to the next step. The whole process is iterated until the termination of the environment simulation. Then the Cart-Pole environment is reset and started again from a new initial state to perform the next episode of the training.



Terminate
Yes
No
Maximum Number of Time Step Reached?
Action Output
Cart-Pole Agent
Cart-Pole Environment
Initialization if it's the first time step
Action Output
Q-Value Comparison
Q_{new}=max(Q_{left}, Q_{right})
Q_{left}, Q_{right}
Reset the Environment
Cart-Pole Simulator
Observation
SNN
Observation
Eligibility Trace
New Weights
Yes
Termination Criteria Reached ?
No
Reward=1
TD (Q_{old}, Q_{new},Reward) $Q_{new} \rightarrow Q_{old}$
TD Error
Weight Update
Q_{new} = 0, Reward=0 for unsuccessful ending

Figure 3.1: A general work flow of the training process using TD-STDP

3.1.2. The Architecture of the SNN

The overall structure of the SNN is illustrated in Figure 3.2. There are only two layers of neurons. The first is the input layer, which is made up of a number of input neurons. The second is the output layer, which is made up of two groups of neurons, each group corresponding to one possible choice of the action. Every input neuron is connected to every output neuron through a synapse.

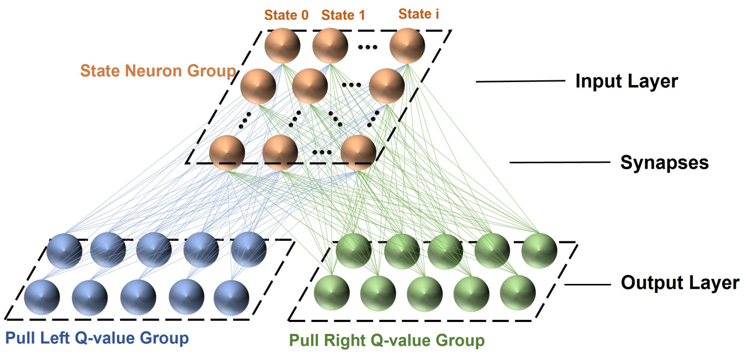


Figure 3.2: The overall structure of the SNN using TD-STDP

3.1.3. 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,

$$id_{obs} = \begin{cases} 0, & obs \leq obs_{min} \\ floor(\frac{x-x_{min}}{\Delta x}), & obs_{min} < obs < obs_{max} \\ N_{states,obs}, -1 & obs \geq obs_{max} \end{cases} \tag{3.1}$$

$$N_{states,obs} = ceil(\frac{x_{max} - x_{min}}{\Delta x}), \tag{3.2}$$

where the variable obs corresponds to one of the observed variables x, v, θ or ω. $[obs_{min}, obs_{max}]$ defines the region 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 input neurons of the SNN is defined to be $N_{states,total} * n_{input}$, so that each possible state is represented by a group of n_{input} neurons. 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 [57].

3.1.4. The Output Neurons

The dynamics of each of the output neurons is described by an LIF model with adaption in Equation 3.3[58].

$$\tau_m \frac{dV}{dt} = g_e(E_e - V) + E_l - V, \tag{3.3}$$

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

$$\tau_g \frac{dg_e}{dt} = -g_e, \tag{3.4}$$

in which τ_e is a time constant for the lasting effect of the input spikes. When there is an input spike from a synapse i connecting to the neuron, g_e is instantaneously changed according to

$$g_e = g_e + w_i, \tag{3.5}$$

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

3.1.5. 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,

$$Q(s,a) = scale * n_{spikes}(s,a), \tag{3.6}$$

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

Note that this definition of the SNN's output as the (scaled) Q-value that depends on both the state and the action is somewhat different from the SNN value network of ref [23]. There, the SNN's output from a single group of neurons provided the values of only the states without further discrimination of different actions.

Hereby using separated groups of output neurons for different actions, we do not need an extra actor module as in ref [23] 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 period of time, obtain the $Q(s_n, a)$ values, choose and carry out 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 2.3, the TD error for step n is computed as

$$TD_n = TD(s_n, a_n) = \gamma max_a Q(s_{n+1}, a) + R_{n+1} - Q(s_n, a_n). \tag{3.7}$$

Here, we have included a "discount factor" γ so that the rewards of only a finite number of future steps have effective contributions 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

$$TD_n = -Q(s_n, a_n). \tag{3.8}$$

As shown in Figure 3.1, the TD error is used to update the weights of the synapses connecting the input state neurons to the output neurons.

3.1.6. 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,

$$A_{pre}(t) = \sum_{k \in input\ spikes} \xi(t - t_k) e^{-\frac{t-t_k}{\tau_{pre}}}, \tag{3.9}$$

$$A_{post}(t) = \sum_{k \in output\ spikes} \xi(t - t_k) e^{-\frac{t-t_k}{\tau_{post}}}, \tag{3.10}$$

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.,

$$\xi(t - t_k) = \begin{cases} 1, & t - t_k \geq 0 \\ 0, & otherwise. \end{cases} \tag{3.11}$$

Clearly, A_{pre} is non-negligible only not long (compared with the time constant t_{pre}) after a pre-synaptic spike has been received. Similarly, A_{post} is non-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

$$w_{trace} = \Delta_{pre} \int O(t) A_{pre}(t) dt - \Delta_{post} \int I(t) A_{post}(t) dt, \tag{3.12}$$

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

$$I(t) = \sum_{k \in input\ spikes} \delta(t - t_k), \tag{3.13}$$

$$O(t) = \sum_{k \in output\ spikes} \delta(t - t_k). \tag{3.14}$$

In Equation 3.12, 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 the receiving of a pre-synaptic 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 non-negligible).

3.1.7. Learning the Synaptic Weights by TD-STDP

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

$$w_{new} = w_{old} + \beta * TD_n * w_{trace,n}, \tag{3.15}$$

where β is a constant that adjusts the rate of the weight change, and $w_{trace,n}$ is the eligibility trace calculated using Equation 3.12 at the nth Cart-Pole step.

3.1.8. Exploration and Exploitation in Training

In order for the agent to accumulate experiences in various Cart-Pole states, the exploration of various possible states should be encouraged 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 episodes, the agent always 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 downscaled at the beginning of each new episode by a factor of $\alpha = 0.99$. With a probability of $1 - P_{explore}$, the agent chooses the action according to

$$P(a_n = a) \propto e^{(Q_a/\delta_{Q_0})}. \tag{3.16}$$

With the above definition, the action of the agent 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).

3.2. The R-STDP SNN

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

The overall framework of the R-STDP program shown in Figure 3.3 is similar to the overall framework of the TD-STDP program. The main differences between the two programs come from the different interpretation 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 more number of spikes is preferred). Thus, upon 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.

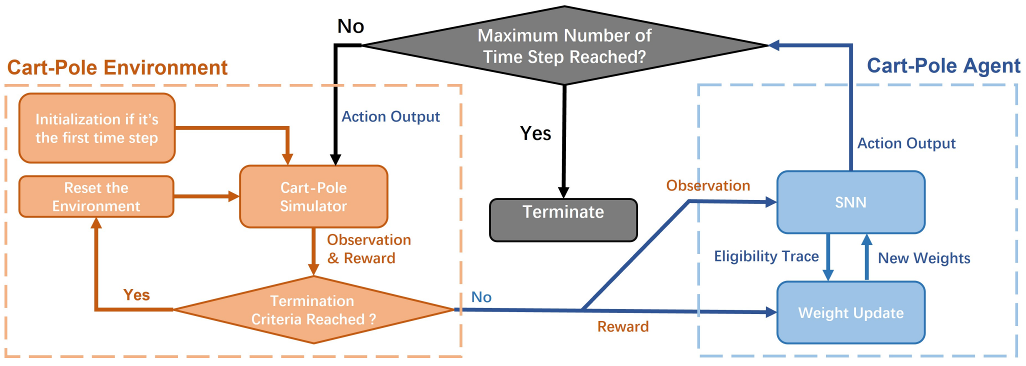


Figure 3.3: A general work flow of the training process using R-STDP

3.2.2. Updating Synapse Weights with Delayed Reward

After the action (chosen based on the output of the SNN) has been taken, a new state of the Cart Pole is returned. A reward value 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 as well as the eligibility factors computed from the previous run of the SNN. When the action taken leads to a positive reward, the synapse connections that would increase the probability of the SNN to output the taken action should be strengthened, while the synapse connections that would increase the probability of the SNN to produce 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. Namely, for every synapse connecting to the output neuron associated with the taken action, the weights are increased according to

$$w_{new} = w_{old} + R(s_n, s_{n+1}) * w_{trace,n}, \tag{3.17}$$

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

$$w_{new} = w_{old} - R(s_n, s_{n+1}) * w_{trace,n}. \tag{3.18}$$

3.2.3. 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 the achieving of the control goal. Based on this intuition, we designed and tested three different reward functions for the SNN controller. From reward function 1 to reward function 3, more and more sensory information is encoded into the reward function, and therefore, the actual tendency of the agent to keep the pole in the upright position is more accurately pictured. In the following formulations, the subscript old corresponds to the Cart-Pole step n in equations 3.17 and 3.18 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 1 when the pole angle is in the range where the pole does not fall and the cart is still within the range of the animation display, and to return 0 otherwise. That is,

$$R_1 = \begin{cases} 1, & The\ simulation\ is\ not\ terminated \\ 0, & otherwise. \end{cases} \tag{3.19}$$

Reward Function 2 This reward function is defined as

$$R_2(\omega_{old}, \omega_{new}) = \begin{cases} 1, & \omega_{old} * \omega_{new} < 0\ or \\ & |\omega_{old}| > |\omega_{new}| \\ -1, & otherwise. \end{cases} \tag{3.20}$$

This reward is determined by comparing the current angular velocity ω_{new} with the angular velocity observed at the previous Cart-Pole step ω_{old}. If a sign change of the angular velocity has been detected, this means that the action has reversed the previous moving trend of the pole, and thus a reward of 1 is assigned to this kind of action. If there is no sign change of the angular velocity, the shrinking of the absolute value of the angular velocity itself also indicates that the pole is slowing down its rotation. The action is thus also encouraged with a reward of 1. 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

$$R_3(\omega_{old}, \omega_{new}, \theta_{old}, \theta_{new}) = \begin{cases} R_2(\omega_{old}, \omega_{new}), & \theta_{new} * \omega_{old} > 0 \\ 1, & \theta_{new} * \omega_{old} \leq 0 \; and \; \theta_{new} * \omega_{new} < 0 \\ -1, & otherwise. \end{cases} \tag{3.21}$$

Compared with the 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 but of opposite signs, θ_{old} must be of the same sign as θ_{new}, because

$$\theta_{old} = \theta_{new} - \omega_{old} * \Delta t. \tag{3.22}$$

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

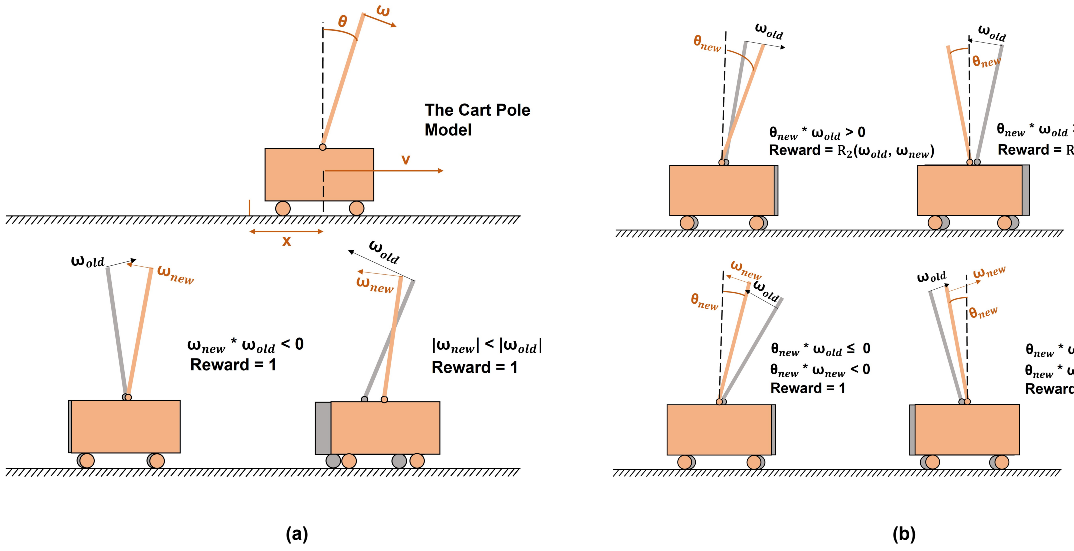


Figure 3.4: Different reward function designs. (a) Reward function design 2 that employs the angular velocity of the pole. (b) Reward function design 3 that employs the angular velocity of the pole as well as the angle of the pole.

3.2.4. Exploitation and Exploration

With the above R-STDP learning rules and the encoding scheme of the Cart-Pole states, 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 those 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 the action determined from the SNN's outputs (exploitation). The value of $P_{explore}$ is set to be 1 at the beginning of the learning and is downscaled by a value of $\gamma_{explore}$ = 0.9 at the start of each new episode, so that as the training process progresses, fewer random actions will be taken and the learning process can eventually converge.

4 Results

In this chapter, we present results of using TD-STDP and R-STDP to solve the Cart-Pole problem. For TD-STDP, we describe a working setup and effects of varying several design choices. For R-STDP, we present the results of using three different designs of the reward function.

4.1. TD-STDP Learning

4.1.1. Results of a Working Setup

The TD-STDP learning process described in the previous chapter involves a range of parameters and design choices, which include the various time constant and voltage parameters of the spiking neurons, the learning rate, and the strategy for exploration/exploitation. For parameters of the SNN, we have mostly used the default values from the Brian2 (see corresponding entries in Table 4.1)

Parameter	Value
τ_m	10 ms
τ_g	5 ms
E_e	0 mV
E_l	-74 mV
$V_{threshold}$	-54 mV
V_{reset}	-60 mV
γ	0.98
τ_{pre}, τ_{post}	20 ms
Δ_{pre}	0.0001
Δ_{post}	0.00001*Δ_{pre}
$N_{states,total}$	120
n_{input}	1
n_{output}	10

Table 4.1: Parameters used for the SNN network. $V_{threshold}$ and V_{reset} are respectively the threshold potential for generating a spike and the reset potential after spike generation for an output neuron. Other symbols are defined in Chapter 3.

After some initial trials and errors, we found that the exploration/exploitation scheme given in Section 3.1.8 and a value of 0.01 for the learning rate β in Equation 3.15 leads to reasonable performance.

During one training session, the duration starting from the initialization/resetting of the Gym Cart-Pole environment to the termination of the environment is considered as one episode. As the goal of the Cart-Pole problem is to keep the pole in the upright position as long as possible, we define an episode to be successful if the number of time steps for which the pole has not fallen down is larger than a threshold of $n_{threshold} = 200$. At a particular training stage, the performance can be evaluated by considering the success rate over 20 consecutive episodes centered around that stage.

Figure 4.1a shows the success rates as functions of the number of training episodes in three independent runs using the above setup.

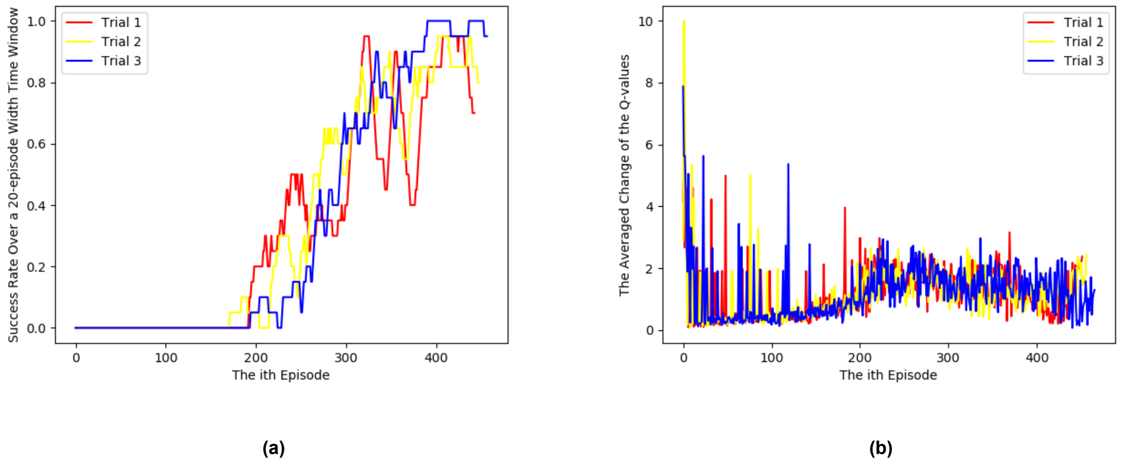


Figure 4.1: The training results of the SNN using TD-STDP, where the three differently colored lines indicate the three independent trials. (a) The success rate of the SNN controller changes over the training episodes using TD-STDP learning. (b) The root mean square change of the SNN-produced Q-values between two consecutive training episodes during TD-STDP training. In this and in the subsequent figures, different colors indicate different and independent training runs.

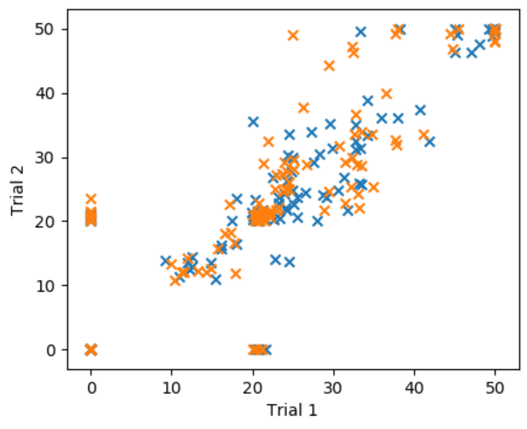
As the training process evolves, the success rate of the SNN controller generally increases, but with several drops. These drops could be the results of the agent encountering states that it does not yet have enough experiences to form a good policy. This change of success rate indicates that the training process is indeed converging, although in a somewhat noisy manner. This phenomenon can also be seen from Figure 4.1b which plots the root mean square change of the Q-values between two consecutive training episodes against the number of training episodes. The curves exhibit fluctuations within a certain range even after prolonged training.

4.1.2. Q-values Learnt in Different Runs

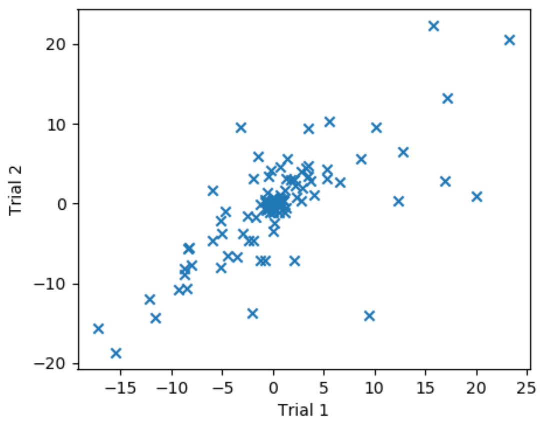
In Figure 4.2, the SNN generated Q-values from two different prolonged runs are compared. Those Q-values were extracted right after the value of 1.0 for the 20-episode-window success rate was reached for the first time in the respective runs.

Figure 4.2a shows that although the Q-values learnt from the two different runs are not exactly the same, they are highly correlated. We note that in this figure, the Q-values of 0 are associated with states that have never been visited in the corresponding run. Obviously, there are a few states that have been visited only in one run but not in the other run. For such states their probabilities of being visited are low and to fill the corresponding entries in the Q-value table with high certainty requires quite extensive training. Visiting such states at the very late stages of training should be the reason for the sporadic occurrences of unsuccessful episodes even after long training processes.

Figure 4.2b indicates that for a given state, the Q-value changes of one action relative to the other are also highly correlated between the two runs. More importantly, the signs of the Q-value changes for most states are conserved across different runs, meaning that independently trained SNN agents will likely choose the same action at the same state. Thus, the SNNs trained by TD-STDP learning have generated meaningful Q-value tables in spite of noises.



(a)



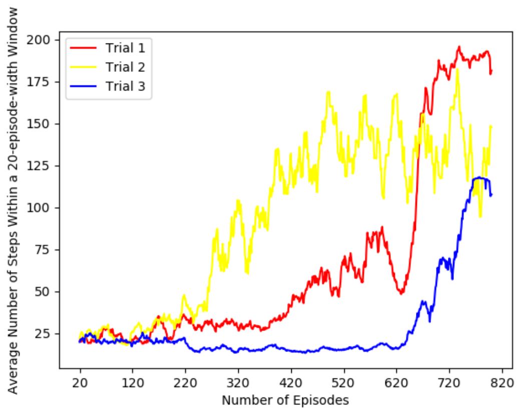
(b)

Figure 4.2: Comparison of Q-values generated by SNNs trained in different runs. (a) Each point in this scatter plot corresponds to a state-action pair of the Cart-Pole agent. Points of different colors correspond to different actions. Values along the horizontal axis are from one run and those along the vertical axis are from the other run. (b) Each point in this figure corresponds to a state, and the scatter plot compares the Q-value change of one action relative to the other at that state from the two different runs.

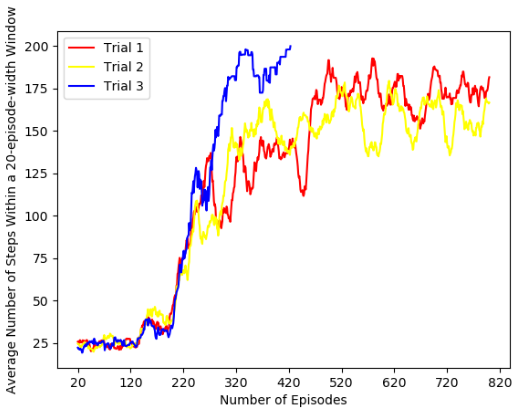
4.1.3. Effects of Varying the Learning Rate

Clearly, the performance of the SNN will depend on the parameters. To explore the entire parameter space requires extensive resources and is out of the scope of the current study. Here we have focused on the effect of changing the learning rate parameter β in Equation 3.15 and the exploration/exploitation scheme.

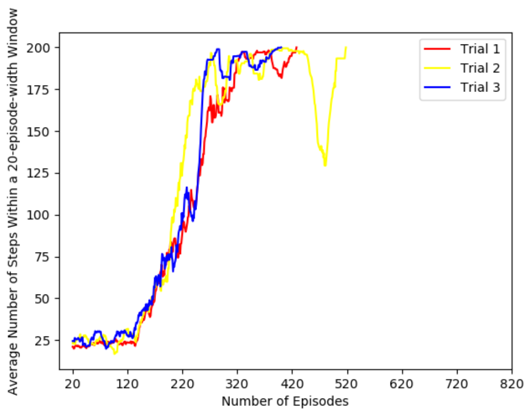
Figure 4.3 and Table 4.2 show the effects of varying the value of β in independent training runs lasting maximally 800 episodes. Since not all learning rate settings could ensure the Cart-Pole agent to achieve any success episode within 800 episodes of training, instead of using success rate, we use here the average steps within a 20-episode-width window as the performance metric in Figure 4.3. Table 4.2 lists the numbers of episodes passed for this metric to first reach different threshold values. The results indicate that while too small learning rates lead to low efficiency of learning (Figure 4.3a, too large learning rates (Figure 4.3e cause the learning to be unstable. The value of 0.01 from our exploratory investigation seems to be a well-balanced choice.



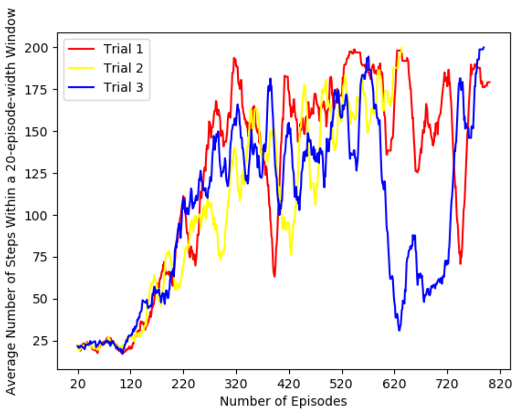
(a)



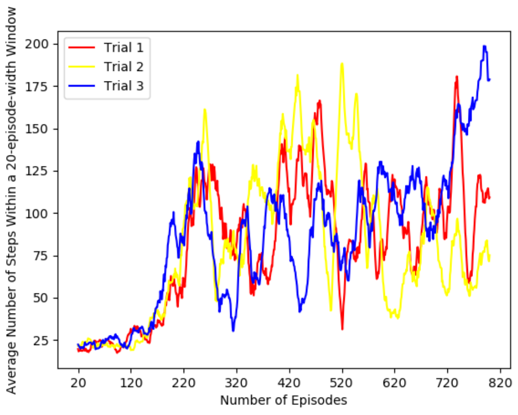
(b)



(c)



(d)



(e)

Figure 4.3: The training results of the SNN using TD-STDP with different learning rate using the average number of sustained Cart-Pole steps in an episode within a 20-episode-width window as the metric. (a)β = 0.001. (b) β = 0.005. (c) β = 0.01. (d) β = 0.05. (e) β = 0.1.

Beta	Number of episodes to reach threshold 101	Number of episodes to reach threshold 176	Number of episodes to reach threshold 196	Number of episodes to reach threshold 200
0.001	(658, 319, 752)	(693, 735, *)	(*, *, *)	(*, *, *)
0.005	(240, 242 ,235)	(488, 515, 303)	(*, *, 326)	(*, *, 423)
0.01	(232, 205, 219)	(294, 243, 258)	(327, 272, 280)	(428, 518, 400)
0.05	(216, 257, 218)	(312, 449, 381)	(535, 632, 781)	(*, 634, 789)
0.1	(232, 223, 227)	(736, 436, 779)	(*, *, 789)	(*, *, *)

Table 4.2: The training results of the SNN using TD-STDP with different learning rates. Different columns list the numbers of episodes passed for the number of sustained Cart-Pole steps averaged over 20-episode-width windows to first reach different threshold values. The three numbers in each entry are from three inpendent runs. The symbol "*" indicates that the corresponding threshold were not reached within 800 episodes of training.

4.1.4. Effects of Varying the Exploration/Exploitation Scheme

The results in Figure 4.4 and Table 4.3 show the effect of using different exploration and exploitation schemes. The schemes considered include the following.

- **Scheme 1**: The scheme described in section 3.1.8, which is to first use 100 episodes of completely random exploration ($p_{explore} = 1.0$) and then downscale $p_{explore}$ by the factor of 0.99 per episode.
- **Scheme 2**: As Scheme 1 but without the starting 100 episodes of completely random exploration.
- **Scheme 3**: No exploration is performed, that is, $p_{explore}$ is set to zero from the beginning of training.
- **Scheme 4**: As Scheme 1 but $p_{explore}$ is immediately changed from 1.0 to 0 after the first 100 episodes.

The schemes with gradually decaying $p_{explore}$ (Figures 4.4a and 4.4b) lead to more robust training than the schemes with the abruptly changed $p_{explore}$ (Figure 4.4d) or without exploration (Figure 4.4c).

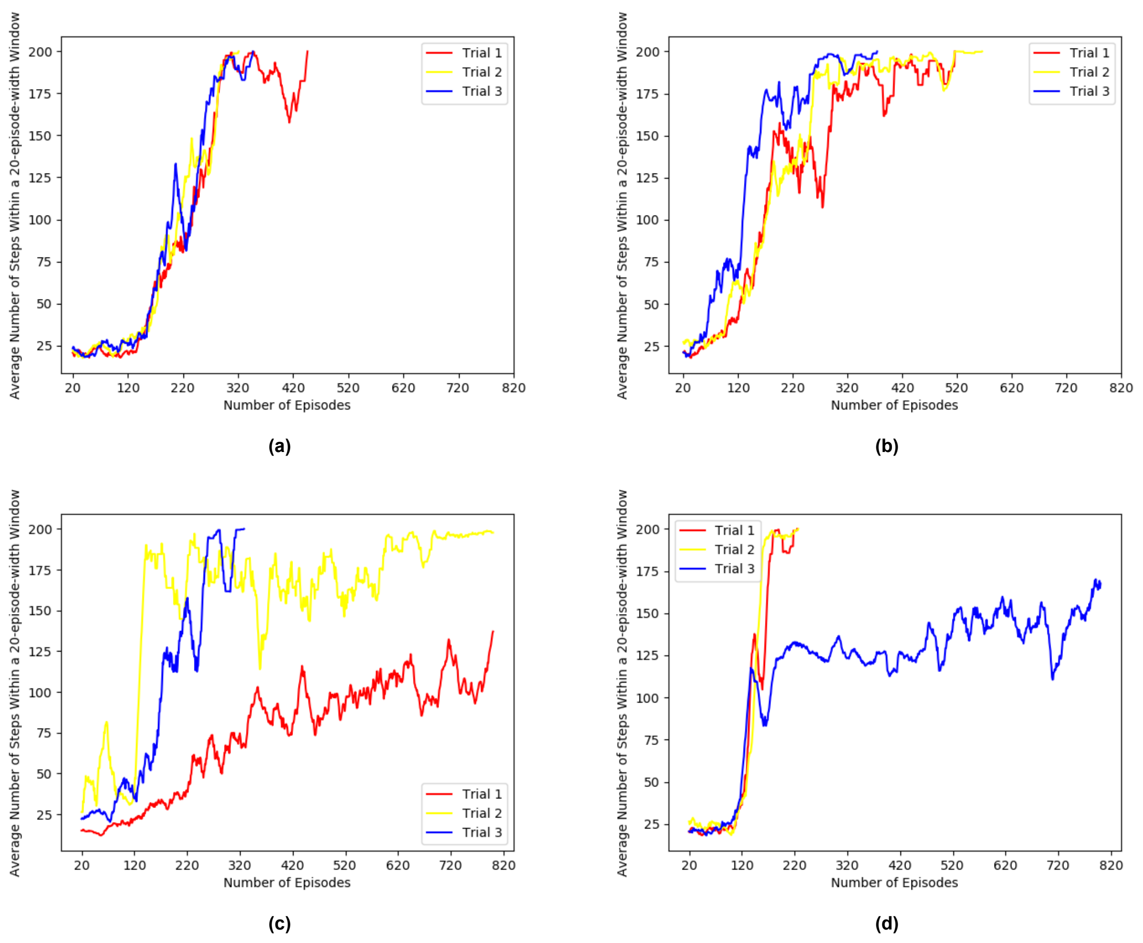


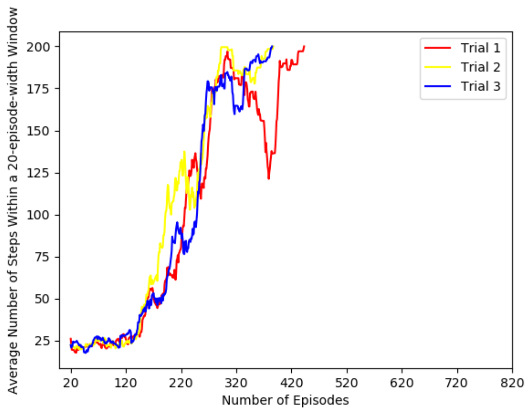
Figure 4.4: The training results of the SNN using TD-STDP with different exploration/exploitation schemes using the average number of sustained Cart-Pole steps within a 20-episode-width window as the metric. (a) Scheme 1. (b) Scheme 2. (c) Scheme 3. (d) Scheme 4.

Exploration Strategy	Number of episodes to reach threshold 101	Number of episodes to reach threshold 176	Number of episodes to reach threshold 196	Number of episodes to reach threshold 200
1	(235, 211, 201)	(286, 284, 271)	(298, 311,303)	(446, 321, 347)
2	(168, 172, 129)	(294, 258, 170)	(435, 306, 277)	(516, 566, 374)
3	(353, 129, 173)	(*, 140, 254)	(*, 234, 273)	(*, *, 328)
4	(136, 144, 133)	(172, 158, *)	(180, 172, *)	(225, 227, *)

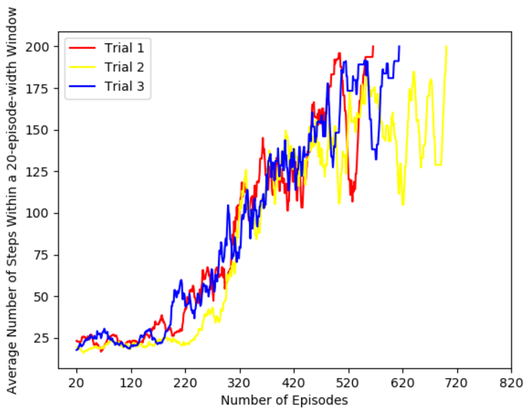
Table 4.3: The same as Table 4.2, but for results of different exploration/exploitation schemes.

4.1.5. The TD Error for Failure State-Action Pairs

We also looked at the effects of not using Equation 3.8, which sets the TD error of a failure state-action pair to the negative of the pair's previous Q-value in the learning process. In Figure 4.5 and Table 4.4, the results obtained by using Equation 3.8 are compared with the results obtained by using the value of 0 for the TD errors of failure state-action pairs. These results indicate that using Equation 3.8 leads to not only more efficiency but also more learnt Q-values to fill in the resulted Q-table.



(a)



(b)

Figure 4.5: The training results of the SNN using TD-STDP with different TD error values for failure state-action pairs using the average number of steps in an episode within a 20-episode-width window as the metric. (a) Using the negative of the current Q values. (b) Using the value of 0.0.

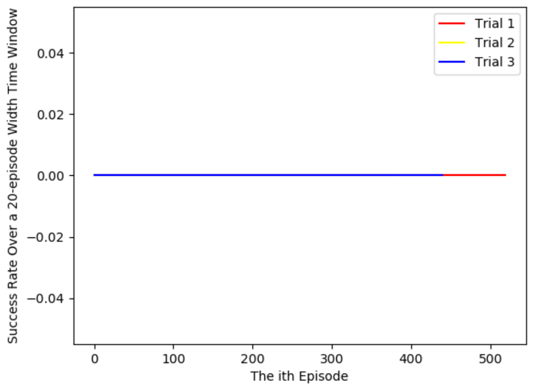
TD Update Strategy	Number of episodes to reach threshold 101	Number of episodes to reach threshold 176	Number of episodes to reach threshold 196	Number of episodes to reach threshold 200
1	(227, 192, 250)	(281, 284, 268)	(304, 292, 382)	(443, 387, 385)
2	(315, 320, 299)	(485, 546, 482)	(566, 701, 614)	(566, 701, 614)

Table 4.4: The same as Table 4.2, but for comparing results of using different TD error values for failure state-action pairs.

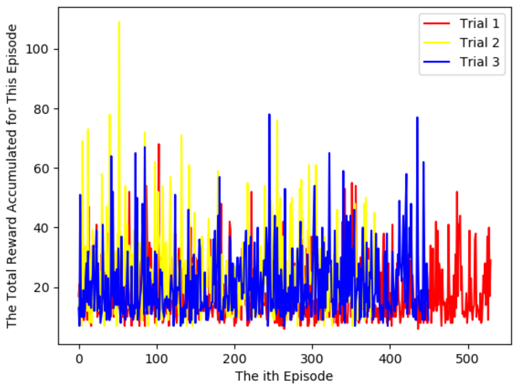
4.2. R-STDP Learning

We have tested the three different reward functions described in Sction 3.2.3.

4.2.1. Results of Reward Function 1



(a)



(b)

Figure 4.6: The training results of the SNN using R-STDP and Reward Function 1. (a) The success rate of the SNN controller changes over the training episodes using R-STDP learning. (b) The total reward accumulated in one episode versus the number of training episodes.

The use of **Reward Function 1** results in a noisy graph that indicates that the number of time steps an episode lasts is entirely random(Figure 4.6). This implies that the agent is not learning anything and therefore cannot making meaningful choices of actions to counter the falling of the pole. This simple reward design seems not to provide enough timing difference that helps the SNN network to learn a proper weight distribution over the synapses.

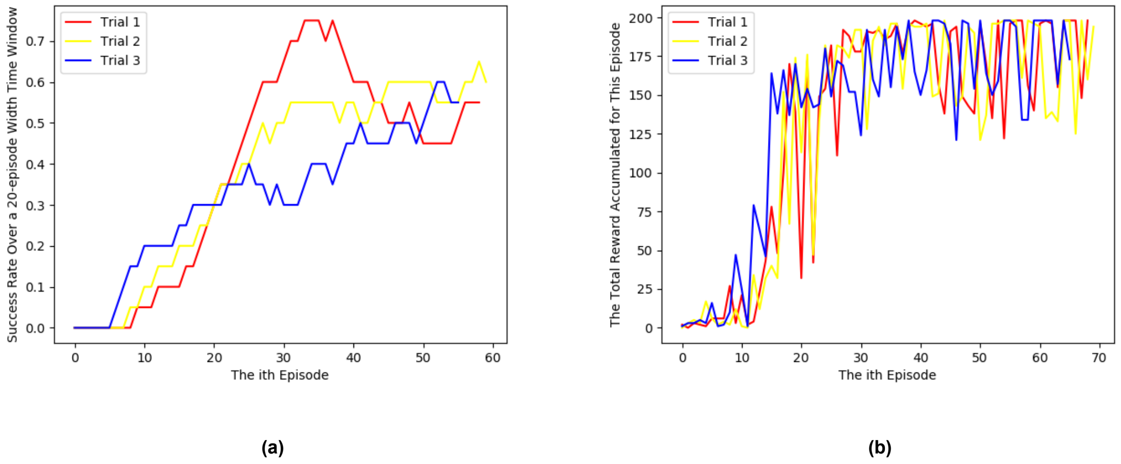


Figure 4.7: The training results of the SNN using R-STDP and Reward Function 2. (a) The success rate of the SNN controller changes over the training episodes using R-STDP learning. (b) The total reward accumulated in one episode versus the number of training episodes.

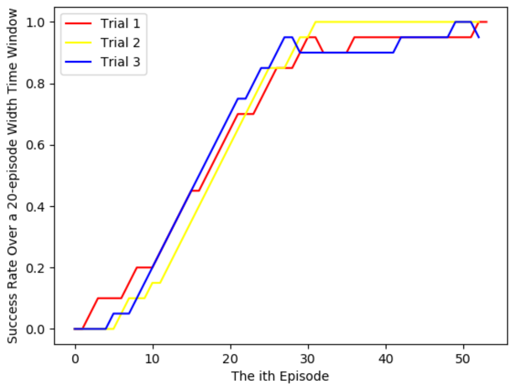
4.2.2. Results of Reward Function 2

As illustrated in Figure 4.7, by using the **Reward Function 2** described in Equation 3.20, the number of time steps that the Cart-Pole simulation is not terminated is generally increasing over the number of episodes. The number of sustained steps eventually fluctuates between 127 and 200, producing success rates between 0.5 to 0.7 which cannot be further improved by longer training. This reward function design only considers the changes of the angular velocity of the pole over two time-steps since the angular velocity represents the tendency of change in terms of the pole angle. However, as this reward function does not take the angular values into consideration, it may actually reward the wrong action choice when the angular value and the angular velocity are of opposite signs.

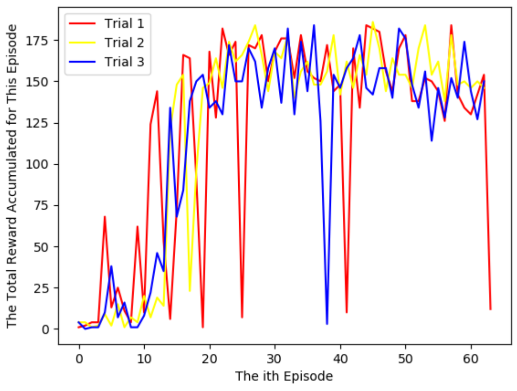
4.2.3. Results of Reward Function 3

Also using the success rate over 20 episodes as the metric for the performance of the SNN at different training stages, the results of R-STDP using **Reward Function 3** (Equation 3.21) are plotted in Figure 4.8a. Figure 4.8a compared with Figure 4.1a, the use of R-STDP shows a more satisfying result. As the training progresses, the success rate increases rather smoothly. They gradually reach the value of 1 with only very small fluctuations in less than 50 episodes. It is interesting to compare this result of R-STDP using **Reward Function 3** with those of TD-STDP, which usually do not show signs of convergence until more than 300 episodes and remained imperfect and noisy until more than 400 episodes.

The accumulated reward in each training episode is shown in Figure 4.8b. As intended, the R-STDP learning process led to a gradual increase in total rewards.



(a)



(b)

Figure 4.8: 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 episodes using R-STDP learning. (b) The total reward accumulated in one episode versus the number of training episodes.

5 Conclusion

Our study has shown that it is feasible to design SNNs for reward optimization to make proper control decisions. Given that STDP learning, the ”native” learning algorithms for SNNs, has so far been relatively under-explored in the context of reinforcement learning for control tasks, the solutions proposed here for the Cart-Pole example can serve as good starting points for developing solutions for more complicated control problems with SNNs implementing the various RL techniques.

The solutions presented here employ a rather simple yet effective SNN structure. By using discrete variables to represent the states of the Cart-Pole and applying one-hot encoding with input spiking neurons, we are able to efficiently achieve the goal of balancing the pole using SNNs of only two layers (i.e., only an input layer and an output layer without any hidden layers).

Although the scheme of one-hot encoding of the input space has proven its effectiveness in this and several previous studies on SNN, the resolution to discretize the state space may affect the performance of the control system. Moreover, as the dimension of the state space increases, the required number of discretized states (and thus the number of input neurons for covering the state space) will increase exponentially. In principle, this problem can be addressed by separately encoding the different input dimensions and then using more hidden layers to project the combined input into some sort of latent representations of the input space. The implementation and testing of such network architectures will be an interesting problem for future investigations.

Given our simple SNN architecture, we investigated two different modulated STDP training methods for RL tasks. TD-STDP is built on the theoretical basis of the 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 episodes, while TD-STDP still delivers noisy result after 300 training episodes), 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 handling 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 crucial in RL control applications. Due to the nature of this type of applications, the reward received by the agent for taking an action is usually delayed relative to the timing of the action, because the feedback of the environment obtainable as the sensory data are usually reactions to previously taken actions. The R-STDP learning rule employed in the training of Cart-Pole controller has successfully used the delayed reward signals to extract useful information to help the SNN generate a stable policy. This is achieved by storing the eligibility traces of the synapses and by using a reward function that is jointly determined by the current and past observations. However, the success of the R-STDP SNN has critically relied on the definition of a reward function that utilized substantial pre-established knowledge or understanding about the dynamics of the problem. This restricts the applicability of the model 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 controller. In this regard, 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.

Despite the fact that RL has been a well-studied field, the training and application of SNN in solving RL problems still remains an active field for ML researchers to explore. An especially intriguing observation is that the RL and the SNN techniques both share the methodology inspired by the learning process of a biological neural system. A long-term goal of this project is to assist the development of control software on neuromorphic computational units. One important follow-up work would be to implement similar SNNs on physical hardware beyond software simulators. The potential of implementing SNN on a hardware level to exploit its power efficiency and the biological relevancy of SNNs compared to the more widely studied ANNs also make SNN a promising approach to further enhance the overall performance of neural network models.

References

[1] Ignacio Abadía et al. “A Cerebellar-Based Solution to the Nondeterministic Time Delay Problem in Robotic Control”. In: *Science Robotics* 6.58 (2021), eabf2756.

[2] Ehsan Arabzadeh, Stefano Panzeri, and Mathew E Diamond. “Deciphering the Spike Train of a Sensory Neuron: Counts and Temporal Patterns in the Rat Whisker Pathway”. In: *Journal of Neuroscience* 26.36 (2006), pp. 9216–9226.

[3] Daniel Auge et al. “A Survey of Encoding Techniques for Signal Processing in Spiking Neural Networks”. In: *Neural Processing Letters* 53.6 (2021), pp. 4693–4710.

[4] Cornelia I Bargmann. “Beyond the Connectome: How Neuromodulators Shape Neural Circuits”. In: *Bioessays* 34.6 (2012), pp. 458–465.

[5] Andrew G Barto. “Reinforcement Learning Control”. In: *Current opinion in neurobiology* 4.6 (1994), pp. 888–893.

[6] Richard Bellman. “Dynamic Programming”. In: *Science* 153.3731 (1966), pp. 34–37. DOI: 10.1126/science.153.3731.34. eprint: https://www.science.org/doi/pdf/10.1126/science.153.3731.34. URL: https://www.science.org/doi/abs/10.1126/science.153.3731.34.

[7] Amine Bermak, Muhammad Hassan, and Xiaofang Pan. “Artificial Olfactory Systems”. In: *Handbook of Biochips*. Springer, 2022, pp. 343–362.

[8] Zhenshan Bing et al. “A Survey of Robotics Control Based on Learning-Inspired Spiking Neural Networks”. In: *Frontiers in neurorobotics* 12 (2018), p. 35.

[9] Sander M Bohte, Joost N Kok, and Johannes A La Poutré. “SpikeProp: Backpropagation for Networks of Spiking Neurons.” In: *ESANN*. Vol. 48. Bruges. 2000, pp. 419–424.

[10] Romain Brette et al. “Simulation of Networks of Spiking Neurons: a Review of Tools and Strategies”. In: *Journal of Computational Neuroscience* 23.3 (2007), pp. 349–398.

[11] Greg Brockman et al. “OpenAI Gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[12] Coleman Brosilow and Babu Joseph. *Techniques of Model-Based Control*. Prentice Hall Professional, 2002.

[13] Nicolas Brunel and Mark CW Van Rossum. “Quantitative Investigations of Electrical Nerve Excitation Treated as Polarization”. In: *Biological Cybernetics* 97.5 (2007), pp. 341–349.

[14] Bruno Cessac, Hélène Paugam-Moisy, and Thierry Viéville. “Overview of Facts and Issues About Neural Coding by Spikes”. In: *Journal of Physiology-Paris* 104.1 (2010). Computational Neuroscience, from Multiple Levels to Multi-level, pp. 5–18. ISSN: 0928-4257. DOI: https://doi.org/10.1016/j.jphysparis.2009.11.002. URL: https://www.sciencedirect.com/science/article/pii/S0928425709000849.

[15] Taylor S. Clawson et al. “Spiking Neural Network (SNN) Control of a Flapping Insect-Scale Robot”. In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. 2016, pp. 3381–3388. DOI: 10.1109/CDC.2016.7798778.

[16] Arnaud Delorme and Simon J Thorpe. “SpikeNET: An Event-Driven Simulation Package for Modelling Large Networks of Spiking Neurons”. In: *Network: Computation in Neural Systems* 14.4 (2003), p. 613.

[17] Jianhao Ding et al. “Optimal ANN-SNN Conversion for Fast and Accurate Inference in Deep Spiking Neural Networks”. In: *arXiv preprint arXiv:2105.11654* (2021).

[18] Jason K Eshraghian et al. “Training Spiking Neural Networks Using Lessons From Deep Learning”. In: *arXiv preprint arXiv:2109.12894* (2021).

[19] Jason Kamran Eshraghian et al. “Training Spiking Neural Networks Using Lessons From Deep Learning”. In: *CoRR* abs/2109.12894 (2021). arXiv: 2109.12894. URL: https://arxiv.org/abs/2109.12894.

[20] Michel Fliess and Cédric Join. “Model-Free Control”. In: *International Journal of Control* 86.12 (2013), pp. 2228–2252.

[21] Răzvan V Florian. “Reinforcement Learning Through Modulation of Spike-Timing-Dependent Synaptic Plasticity”. In: *Neural computation* 19.6 (2007), pp. 1468–1502.

[22] Nicolas Frémaux and Wulfram Gerstner. “Neuromodulated Spike-Timing-Dependent Plasticity, and Theory of Three-Factor Learning Rules”. In: *Frontiers in neural circuits* 9 (2016), p. 85.

[23] Nicolas Frémaux, Henning Sprekeler, and Wulfram Gerstner. “Reinforcement Learning Using a Continuous Time Actor-Critic Framework with Spiking Neurons”. In: *PLoS computational biology* 9.4 (2013), e1003024.

[24] Fabrizio Gabbiani and W Metzner. “Encoding and Processing of Sensory Information in Neuronal Spike Trains”. In: *Journal of Experimental Biology* 202.10 (1999), pp. 1267–1279.

[25] Jacques Gautrais and Simon Thorpe. “Rate Coding versus Temporal Order Coding: a Theoretical Approach”. In: *Biosystems* 48.1-3 (1998), pp. 57–65.

[26] Wulfram Gerstner et al. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014.

[27] Shlomo Geva and Joaquin Sitte. “A Cartpole Experiment Benchmark for Trainable Controllers”. In: *IEEE Control Systems Magazine* 13.5 (1993), pp. 40–51.

[28] Samanwoy Ghosh-Dastidar and Hojjat Adeli. “Spiking Neural Networks”. In: *International journal of neural systems* 19.04 (2009), pp. 295–308.

[29] Alessandro Giusti et al. “A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots”. In: *IEEE Robotics and Automation Letters* 1.2 (2016), pp. 661–667. DOI: 10.1109/LRA.2015.2509024.

[30] Wenzhe Guo et al. “Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems”. In: *Frontiers in Neuroscience* 15 (2021), p. 638474.

[31] Donald Olding Hebb and Wilder Penfield. “Human Behavior After Extensive Bilateral Removal from the Frontal Lobes”. In: *Archives of Neurology & Psychiatry* 44.2 (1940), pp. 421–438.

[32] Morris W Hirsch, Stephen Smale, and Robert L Devaney. *Differential Equations, Dynamical Systems, and an Introduction to Chaos*. Academic press, 2012.

[33] Eugene M Izhikevich. “Which Model to Use for Cortical Spiking Neurons?” In: *IEEE transactions on neural networks* 15.5 (2004), pp. 1063–1070.

[34] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. “Artificial Neural Networks: A Tutorial”. In: *Computer* 29.3 (1996), pp. 31–44.

[35] Tammy Jiang, Jaimie L Gradus, and Anthony J Rosellini. “Supervised Machine Learning: A Brief Primer”. In: *Behavior Therapy* 51.5 (2020), pp. 675–687.

[36] Roland S Johansson and Ingvars Birznieks. “First Spikes in Ensembles of Human Tactile Afferents Code Complex Spatial Fingertip Events”. In: *Nature neuroscience* 7.2 (2004), pp. 170–177.

[37] M. I. Jordan and T. M. Mitchell. “Machine learning: Trends, perspectives, and prospects”. In: *Science* 349.6245 (2015), pp. 255–260. DOI: 10.1126/science.aaa8415. eprint: https://www.science.org/doi/pdf/10.1126/science.aaa8415. URL: https://www.science.org/doi/abs/10.1126/science.aaa8415.

[38] R Kabilan and N Muthukumaran. “A Neuromorphic Model for Image Recognition using SNN”. In: *2021 6th International Conference on Inventive Computation Technologies (ICICT)*. IEEE. 2021, pp. 720–725.

[39] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. “Reinforcement Learning: A Survey”. In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[40] Seijoon Kim et al. “Spiking-Yolo: Spiking Neural Network for Energy-Efficient Object Detection”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 07. 2020, pp. 11270–11277.

[41] Shruti R Kulkarni et al. “Benchmarking the Performance of Neuromorphic and Spiking Neural Network Simulators”. In: *Neurocomputing* 447 (2021), pp. 145–160.

[42] Yann Lecun. “A Theoretical Framework for Back-Propagation”. English (US). In: *Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA*. Ed. by D. Touretzky, G. Hinton, and T. Sejnowski. Morgan Kaufmann, 1988, pp. 21–28.

[43] Eimantas Ledinauskas et al. “Training Deep Spiking Neural Networks”. In: *arXiv preprint arXiv:2006.04436* (2020).

[44] Irwin B Levitan, Irwin B Levitan, Leonard K Kaczmarek, et al. *The Neuron: Cell and Molecular Biology*. Oxford University Press, USA, 2002.

[45] Yuxiang Liu and Wei Pan. “Spiking Neural-Networks-Based Data-Driven Control”. In: *Electronics* 12.2 (2023), p. 310.

[46] Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. “A History of Spike-Timing-Dependent Plasticity”. In: *Frontiers in synaptic neuroscience* 3 (2011), p. 4.

[47] Timothée Masquelier, Rudy Guyonneau, and Simon J Thorpe. “Competitive STDP-Based Spike Pattern Learning”. In: *Neural computation* 21.5 (2009), pp. 1259–1276.

[48] Carver Mead. “Neuromorphic Electronic Systems”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1629–1636.

[49] Signe Moe, Anne Marthine Rustad, and Kristian G. Hanssen. “Machine Learning in Control Systems: An Overview of the State of the Art”. In: *Artificial Intelligence XXXV*. Ed. by Max Bramer and Miltos Petridis. Cham: Springer International Publishing, 2018, pp. 250–265. ISBN: 978-3-030-04191-5.

[50] Sohrab Mokhtari et al. “A Machine Learning Approach for Anomaly Detection in Industrial Control Systems Based on Measurement Data”. In: *Electronics* 10.4 (2021). ISSN: 2079-9292. DOI: `10.3390/electronics10040407`. URL: `https://www.mdpi.com/2079-9292/10/4/407`.

[51] Anne-Marie M Oswald, Brent Doiron, and Leonard Maler. “Interval coding. I. Burst Interspike Intervals as Indicators of Stimulus Intensity”. In: *Journal of neurophysiology* 97.4 (2007), pp. 2731–2743.

[52] Darsh Parekh et al. “A Review on Autonomous Vehicles: Progress, Methods and Challenges”. In: *Electronics* 11.14 (2022). ISSN: 2079-9292. DOI: `10.3390/electronics11142162`. URL: `https://www.mdpi.com/2079-9292/11/14/2162`.

[53] Balint Petro, Nikola Kasabov, and Rita M. Kiss. “Selection and Optimization of Temporal Spike Encoding Methods for Spiking Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.2 (2020), pp. 358–370. DOI: `10.1109/TNNLS.2019.2906158`.

[54] Fred Rieke et al. *Spikes: Exploring the Neural Code*. MIT press, 1999.

[55] Sebastian Ruder. “An Overview of Gradient Descent Optimization Algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).

[56] Bodo Rueckauer and Shih-Chii Liu. “Conversion of Analog to Spiking Neural Networks Using Sparse Temporal Coding”. In: *2018 IEEE international symposium on circuits and systems (ISCAS)*. IEEE. 2018, pp. 1–5.

[57] Cedric Seger. *An Investigation of Categorical Variable Encoding Techniques in Machine Learning: Binary Versus One-Hot and Feature Hashing*. 2018.

[58] Sen Song and Larry F Abbott. “Cortical Development and Remapping Through Spike Timing-Dependent Plasticity”. In: *Neuron* 32.2 (2001), pp. 339–350.

[59] Sen Song, Kenneth D Miller, and Larry F Abbott. “Competitive Hebbian Learning Through Spike-Timing-Dependent Synaptic Plasticity”. In: *Nature neuroscience* 3.9 (2000), pp. 919–926.

[60] Marcel Stimberg, Romain Brette, and Dan FM Goodman. “Brian 2, An Intuitive and Efficient Neural Simulator”. In: *eLife* 8 (2019), e47314.

[61] Clarence Tan, Marko Šarlija, and Nikola Kasabov. “Spiking Neural Networks: Background, Recent Development and the NeuCube Architecture”. In: *Neural Processing Letters* 52 (Oct. 2020). DOI: 10.1007/s11063-020-10322-8.

[62] Amirhossein Tavanaei and Anthony Maida. “BP-STDP: Approximating Backpropagation Using Spike Timing Dependent Plasticity”. In: *Neurocomputing* 330 (2019), pp. 39–47.

[63] Simon Thorpe, Denis Fize, and Catherine Marlot. “Speed of Processing in the Human Visual System”. In: *nature* 381.6582 (1996), pp. 520–522.

[64] Simon Thorpe and Jacques Gautrais. “Rank Order Coding”. In: *Computational neuroscience*. Springer, 1998, pp. 113–118.

[65] Jilles Vreeken et al. “Spiking Neural Networks, An Introduction”. In: (2003).

[66] John J. Wade et al. “SWAT: A Spiking Neural Network Training Algorithm for Classification Problems”. In: *IEEE Transactions on Neural Networks* 21.11 (2010), pp. 1817–1830. DOI: 10.1109/TNN.2010.2074212.

[67] Shouyi Wang, Wanpracha Chaovalitwongse, and Robert Babuska. “Machine Learning Algorithms in Bipedal Robot Control”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.5 (2012), pp. 728–743. DOI: 10.1109/TSMCC.2012.2186565.

[68] Christopher JCH Watkins and Peter Dayan. “Q-Learning”. In: *Machine learning* 8.3 (1992), pp. 279–292.

[69] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3 (1992), pp. 279–292.

[70] Yujie Wu et al. “Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks”. In: *Frontiers in neuroscience* 12 (2018), p. 331.

[71] Zhe Wu et al. “Machine Learning-Based Predictive Control of Nonlinear Processes. Part I: Theory”. In: *AIChE Journal* 65.11 (2019), e16729.

[72] Jian Yin et al. “High-Dimensional Shared Nearest Neighbor Clustering Algorithm”. In: *International Conference on Fuzzy Systems and Knowledge Discovery*. Springer. 2005, pp. 494–502.

[73] Jinming Zou, Yi Han, and Sung-Sau So. “Overview of Artificial Neural Networks”. In: *Artificial Neural Networks* (2008), pp. 14–22.