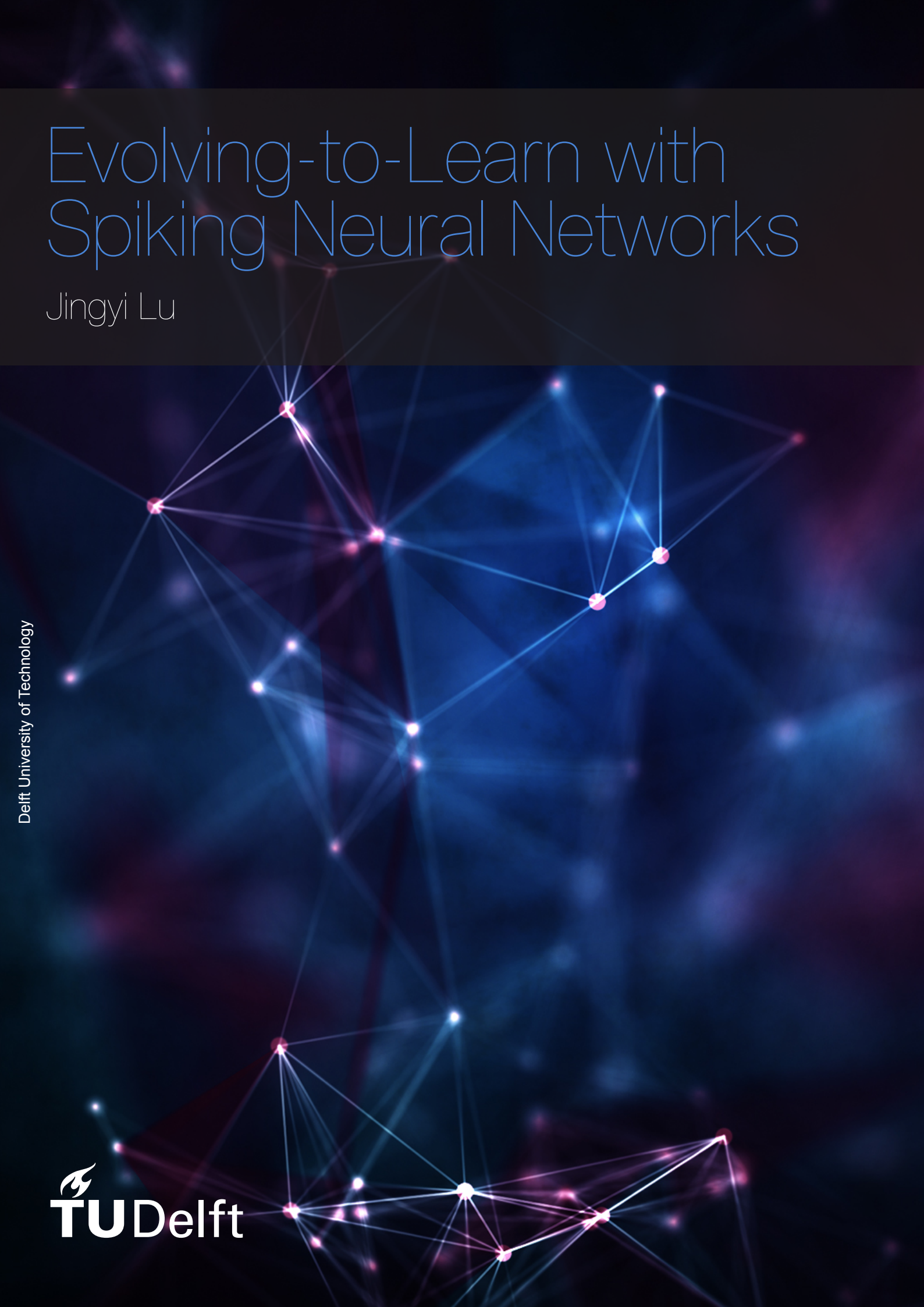
Evolving-to-Learn with Spiking Neural Networks

Jingyi Lu



Delft University of Technology

TU Delft

Evolving-to-Learn with Spiking Neural Networks

by

Jingyi Lu

to obtain the degree of Master of Science
at the Delft University of Technology.

Student number:	4532554	
Project duration:	December, 2020 – January, 2022	
Committee:	Prof. dr. G.C.H.E de Croon,	TU Delft, supervisor
	J.J.Hagenaars MSc,	TU Delft, supervisor
	Dr. O.A. Sharpanskykh,	TU Delft
	Ir. C. de Wagter,	TU Delft
	Dr. J.J.G. Dupeyroux,	TU Delft

TU Delft

Acknowledgements

I would like to express my sincere gratitude to my supervisors Guido de Croon and Jesse Hagenaars. Thank you for your guidance and inspiring advice, I learned a lot during the meetings with you. Besides, your encouragement and support through the past year is of great value to me, which motivates me to overcome difficulties and keep on improving.

I would also like to thank my friends for all the warm support. My dear Aero Girls, thank you for the company in the past five years. The pandemic period was a challenge to all of us, but I'm grateful that we are always by each other's side either physically or virtually.

Finally, I would like to give my thanks to my family. Thank you for encouraging me to discover the world and learn things that I like. Mom, thank you for the spiritual guidance throughout my life, I'm so blessed to have you as my mother.

Jingyi Lu
Delft, January 2022

Abstract

Inspired by the natural nervous system, synaptic plasticity rules are applied to train spiking neural networks. Different from learning algorithms such as propagation and evolution that are widely used to train spiking neural networks, synaptic plasticity rules learn the parameters with local information, making them suitable for online learning on neuromorphic hardware. However, when such rules are implemented to learn different new tasks, they usually require a significant amount of work on task-dependent fine-tuning. This thesis aims to make this process easier by employing an evolutionary algorithm that evolves suitable synaptic plasticity rules for the task at hand. More specifically, we provide a set of various local signals, a set of mathematical operators, and a global reward signal, after which a Cartesian genetic programming process finds an optimal learning rule from these components. In this work, we first test the algorithm in basic binary pattern classification tasks. Then, using this approach, we find learning rules that successfully solve an XOR and cart-pole task, and discover new learning rules that outperform the baseline rules from literature.

Nomenclature

α, β	Rate constants
ΔW	Weight update
Δ_t	Time difference between the pre- and postsynaptic spikes
	Learning rate
λ	Number of off-springs
μ	Number of parents
τ_+, τ_-	STDP time constants
τ_z	Decay constant of the synapse eligibility trace
ξ	Synapse dynamics
A, B, C, D	Coefficients of the parameterized Hebbian rule
A_+, A_-	STDP amplitude
a_d	Learning window of the STDP rule
C_m	Neuron cell capacitance
$D_{T_j}^{tr}$	Support dataset
D_{train}	Training dataset
E	Voltage source of ion flux
E_m	Equilibrium membrane potential
F	Synapse strength
f	classifier function
f_θ	Neural network model with the connection weights θ
f_i	Postsynaptic spike
f_j	Presynaptic spike
g	Nonlinear conductance of the ion channel
$H(t)$	Heaviside function
I	Current
k_θ	Similarity kernel
L	Loss function
m, n, h	Gating variables of the ion channels
N	Evolution population
P^+, P^-	Pre- and postsynaptic traces

P_θ	Probability distribution
R	Environment reward at each episode
r	Reward signal
R_{base}	Baseline environment reward at each episode
R_m	Membrane resistance
$S(t)$	Spike train
T	Duration of network simulation
t_i	Spike time of postsynaptic neuron
t_j	Spike time of presynaptic neuron
u	Membrane recovery variables
u_{rest}	Membrane rest potential
V	Membrane potential
w_{ij}	Connection weights of synapse
x_i, y_i	Input data and label
z	Synapse eligibility trace

Acronyms

ANN artificial neural network.
BNF Backus-Naur form.
BVP Bon-hoeffer-van der Pol.
CGP cartesian genetic programming.
CPPN compositional pattern producing network.
DSL domain-specific language.
EA evolutionary algorithm.
eSNN evolving spiking neural networks.
GE grammatical evolution.
GEP gene expression programming.
HH Hodgkin-Huxley.
I&F integrated-and-fire.
l2l learning-to-learn.
LIF leaky integrated-and-fire.
LTD long-term depression.
LTP long-term potentiation.
MAML model-agnostic meta-learning.
MANN memory-augmented neural network.
MNIST Modified National Institute of Standards and Technology.
MSTDP modulated STDP.
MSTDPET modulated STDP with eligibility trace.
NEAT neuroevolution of augmenting topologies.
PPO proximal policy optimization.
ReSuMe Remote supervised method.
RNN recurrent neural network.
RSTDP reward-modulated STDP.
SNN spiking neural network.
SPAN spike pattern association neuron.
STDP spike-timing-dependent-plasticity.
TD-STDP temporal difference modulated STDP.
XOR exclusive OR.

List of Figures

List of Tables

Contents

1

Introduction

Throughout the history of artificial intelligence, scientists and researchers have been striving for imitating human brains. Human brains can perform complex tasks with extremely low power consumption. The signal transmission in human brains is performed by electrical signals in the form of spike trains, which is considered to be an important reason for the low consumption. Inspired by it, neuromorphic hardware containing analog circuits has been invented (Mead, 1990). In recent decades, various neuromorphic chips have been developed such as the TrueNorth from IBM and Loihi from Intel (Modha, 2014; Davies et al., 2018). With the development of neuromorphic chips, research in spiking neural networks (SNNs) which can be implemented in neuromorphic chips has become a popular research topic in the field of artificial intelligence. Compared to conventional artificial neural networks (ANNs), the working principle of SNNs is closer to that of the natural nervous systems. Regarding the neural network architecture, a component usually ignored by conventional ANNs is the model of each neuron. While in SNNs, biological neural models are implemented in individual neurons, where information is only transmitted by the neuron when its membrane potential reaches the threshold of a spike (Gerstner et al., 2014). Another major difference between conventional ANNs and SNNs is the format of real information transmitted. Information transmission in ANNs is usually in the format of values and involves large-scale and dense matrix operations. However, SNNs encode values to sparse spike trains containing both spatial and temporal information. These unique features provide SNNs the potential to be significantly more energy-efficient than conventional ANNs (Tavanaei et al., 2019), thus, favorable for applications in power-constrained systems such as robotics and micro air vehicles.

Although the advantage of SNNs in reducing energy has been proved both theoretically and experimentally (Merolla et al., 2014; Poon and Zhou, 2011), a critical problem of lacking efficient algorithms for training SNNs remains unsolved, which limits the learning capability of SNNs. For training ANNs, commonly used efficient learning algorithms are usually based on the backpropagation method, where the neural networks are learned by computing the gradient of the loss function with respect to the weights. However, the backpropagation algorithm fails to work as efficiently in SNNs due to the complex neuron models and discrete spiking signals (Lee et al., 2016). Currently, SNNs are usually trained using synaptic plasticity rules. Synaptic plasticity rules are learning rules based on the synapse connection scheme discovered in biological nervous systems. They describe the strengthening or weakening of synapse connection for pairs of pre- and postsynaptic neurons (Markram et al., 1997). When the synaptic plasticity rules are used for training SNNs, the connection weights of the neural network are represented by synapse connection strength. These learning rules can not only train SNNs to perform pattern classification tasks without labels but also perform supervised learning tasks by introducing teaching signals (Diehl and Cook, 2015; Ruf and Schmitt, 1997). Later research also proved that SNNs can be trained to perform reinforcement learning tasks with reward modulated plasticity rules (Florian, 2007). Aside from learning with plasticity rules, evolutionary algorithms have also been implemented for training SNNs recently (Hagenaars et al., 2020).

Different from the approaches mentioned above, the proposed project will apply the idea of meta-learning to the training process of SNNs, which allows an automated search of learning rules. Meta-

learning is a broad field referring to the learning-to-learn in artificial intelligence (Schmidhuber, 1987). The goal of meta-learning is to allow the neural networks to learn new tasks efficiently and therefore, improve the learning capability of the learning algorithms. The idea of meta-learning has been extended to the field of SNNs for optimizing the learning process (Jordan et al., 2021; Risi and Stanley, 2010). Compared to training SNNs with synaptic plasticity rules on a specific task, the meta-learning approach allows the learning rules to adjust dynamically based on the task family the network is trained on. It can improve the generality and flexibility of the algorithm. Additionally, the process of searching learning rules has the potential to unravel the mechanisms of synaptic plasticity in biological neural systems, leading to the discovery of new effective learning rules.

1.1 Motivation and Research Question

As discussed before, the potential of decreasing power consumption makes SNNs ideal for low-power applications such as micro-robotics and drones (Tavanaei et al., 2019). The development of neuromorphic hardware such as the Loihi chip also encourages the implementation of SNNs (Davies et al., 2018). However, the major obstacle for wide applications of SNNs is the lack of effective learning algorithms. Although synaptic plasticity-based learning algorithms are commonly used for training SNNs, the learning rules and implementation details are not universal. Original synaptic plasticity rules were derived based on discoveries in biological neuron behaviours. However, they are usually modified based on the task and network characteristics. For example, Guo et al. (2019) applied a softbound to the basic STDP rule while Diehl and Cook (2015) implemented a triplet STDP rule. The derivation of plasticity rules for a specific task usually requires a large amount of work. Additionally, there is a lack of comparison between alternative learning rules for solving a certain type of task since an exhaustive comparison is also time-consuming (Jordan et al., 2021). Thus, the goal of this research is to enhance the learning capability of SNNs by developing an algorithm that evolves learning rules automatically. The research objective is phrased as follows:

To achieve effective learning rules using local information for spiking neural networks by means of evolutionary programming.

To realize the research objective, various questions need to be answered. The expression method of learning rules and the evolutionary algorithm are the major problems to be solved. The type of task that the SNNs will be trained on is selected to be continuous control problems since they are essential applications of neural networks on robotics and they are relatively more complex than simple pattern recognition tasks such as MNIST tasks. The main research question is then formulated:

How can effective learning rules be evolved for training a predefined spiking neural network on continuous control problems?

To answer the main research question, several sub-questions need to be studied. Since the subjects to be evolved are learning rules, how to describe the learning rule should be figured out. Additionally, the input signals required for the evolutionary search and an appropriate evolution strategy should be investigated. Thus, the main research question is divided into the following sub-questions:

- **How can the learning rules be expressed?**
- **What are the input signals needed for the evolutionary search?**
- **What is an appropriate evolution strategy for the evolutionary search?**

1.2 Structure of the Report

The report is structured into three parts. Part II presents an overview of the literature on the related topics. Starting from an introduction to the SNNs, the background knowledge in neuroscience, neuron models, and general learning approaches are discussed. Then, the information on meta-learning is presented including general categories of meta-learning and implementations in SNNs. A summary of the information is given in the end of the literature review.

Part III discusses the preliminary experiments of the thesis. To obtain a better understanding of the topic and analyze alternative solutions, in a total of three experiments with different settings were performed. The methodology of the preliminary experiments is provided, where the logic flow between the experiments and the settings for each experiment are given. The results for each experiment are demonstrated afterward. Finally, the analysis of the preliminary experiments is presented.

The appendix is attached in the end as Part III.

I

Scientific Paper

Evolving-to-Learn Reinforcement Learning Tasks with Spiking Neural Networks

J. Lu* , **J. J. Hagenaars** , **G. C. H. E. de Croon**
Micro Air Vehicle Laboratory, Delft University of Technology

Abstract

Inspired by the natural nervous system, synaptic plasticity rules are applied to train spiking neural networks with local information, making them suitable for online learning on neuromorphic hardware. However, when such rules are implemented to learn different new tasks, they usually require a significant amount of work on task-dependent fine-tuning. This paper aims to make this process easier by employing an evolutionary algorithm that evolves suitable synaptic plasticity rules for the task at hand. More specifically, we provide a set of various local signals, a set of mathematical operators, and a global reward signal, after which a Cartesian genetic programming process finds an optimal learning rule from these components. Using this approach, we find learning rules that successfully solve an XOR and cart-pole task, and discover new learning rules that outperform the baseline rules from literature.

1 Introduction

In the 1990s, research progress in the field of neural dynamics in biological nervous systems promoted the development of spiking neural networks (SNNs) [Maas, 1997]. SNNs are a special type of artificial neural network (ANN) composed of biologically more realistic computational units. Similar to the communication between biological neurons, information transfer in SNNs is performed via spikes. A spike is a discrete event that occurs when the membrane potential of a neuron exceeds the threshold [Gerstner *et al.*, 2014]. As a result, neurons in SNNs are sparsely activated, giving these networks the potential to perform computations more energy-efficiently compared to conventional ANNs, which communicate with dense, continuous values [Maas, 1997; Stone, 2016]. For this reason, SNNs are promising for power-limited applications. Additionally, the recent development in neuromorphic chips has paved the way for the hardware implementation of SNNs [Davies *et al.*, 2018; DeBole *et al.*, 2019], and they have been trained successfully with methods such as evolution [Howard and Elfes, 2014; Hagenaars *et al.*, 2020] or backpropagation of surrogate gradients [Shrestha and Orchard, 2018; Neftci *et al.*, 2019]. Nevertheless, these algorithms, making use of a lot of non-local information, are not particularly suitable for on-chip learning in neuromorphic hardware.

Efficient on-chip learning of SNNs can be realized through synaptic plasticity rules like Hebb's rule [Hebb, 1949] or spike timing-dependent plasticity (STDP) [Markram *et al.*, 1997], or improved variants like Oja's rule [Oja, 1982] and triplet STDP [Pfister and Gerstner, 2006]. Some of these learning rules were derived from experimental data [Artola *et al.*, 1990], while others were derived to optimize some metric, like the efficiency of information transmission [Toyoizumi *et al.*, 2005]. Most research takes both factors into account in the derivation process of the learning rule, but how much each aspect is relied on is rarely specified. Additionally, the discovery of alternative learning rules is usually a procedure of trial-and-error, which can be a time-consuming task. To solve these issues, Jordan et al. [2021] proposed an evolutionary algorithm to discover the mathematical expressions of synaptic plasticity rules. Genetic programming was applied to evolve the rules to train SNNs on diverse tasks. The evolutionary search successfully (re)discovered existing solutions with high performance, and identified the essential terms for training. In some tasks, the algorithm also evolved new learning rules with competitive performance. However, since this work was the first attempt to evolve symbolic synaptic plasticity rules for SNNs, the tasks selected for the training were relatively simple and the search space was confined to the components present in existing solutions.

As an extension of [Jordan *et al.*, 2021], we will implement the same algorithm to evolve synaptic plasticity rules for training SNNs, targeting reinforcement learning tasks with increased task complexity and an expanded evolutionary search space. The objective is to evolve synaptic plasticity rules for solving reinforcement tasks, identify the critical terms for successful learning, and discover new learning rules with comparable or better performance.

This paper will start with an overview of related work and the methodology in Section 2 and Section 3, respectively. Next, Section 4 will present the results of the experiments. We will end with a discussion of the experiments and results in Section 5.

*Contact Author

2 Related Work

The idea of evolving synaptic plasticity rules was first applied to ANNs. To improve the learning capability of ANNs with biologically plausible models, Bengio et al. [1991] optimized the parameters of synaptic plasticity rules using gradient descent and a genetic algorithm. Later, the generality of this approach was demonstrated by applying it to learn simple but diverse tasks: a biological circuit, a Boolean function, and various classification tasks [Bengio *et al.*, 2007]. Following a similar approach, Niv et al. [2002] and Stanley et al. [2003] evolved learning rules to train ANNs to perform reinforcement learning tasks and automatic control tasks with adaptive environments, respectively. In all these works, the optimization of the learning rules is based on a parametric function, which is usually an existing synaptic plasticity rule such as Hebb's rule. The optimizer only learns the coefficients of the function. The main advantage of this approach is that fixing the format of the learning rule constrains the search space and reduces the computational effort. However, the robustness and generality of this approach are compromised since variables outside the parametric function are all excluded.

A more general method is to replace the symbolic learning rule with an ANN and optimize it for better learning performance. The adaptive HyperNeat algorithm is a meta-learning method that evolves the connection pattern and weights of the ANN which encodes the learning rule [Risi and Stanley, 2010]. This approach uses neuronal states such as neuron traces and spike activities as inputs to the ANN, which outputs the weight update of the synapse in question. Recently, a similar approach has been implemented in the neuromorphic hardware to optimize the training of SNNs in basic reinforcement learning tasks [Bohnstingl *et al.*, 2019]. This approach simplifies the adaptive HyperNeat algorithm and only evolves the connection weights of the ANN. Both works compared their approaches with methods that optimize a parametric learning rule on the same tasks, and it was found that optimizing the learning rule encoded by an ANN is more robust to tasks with changing environments. The drawback of this approach is also obvious. Since the learning rules are expressed by ANNs, which are hardly interpretable, it is difficult to analyze the learning behaviour and generalize the evolved learning rules to learn other tasks.

With the development of computational neuroscience and SNNs, various synaptic plasticity rules have been derived for learning diverse tasks [Markram *et al.*, 1997; Florian, 2017]. To come up with more general rules that can be applied to a broader array of problems, researchers have proposed meta-learning approaches that evolve and evaluate the symbolic expressions of the synaptic plasticity rules. Different from the previous methods that have a fixed function, the coefficients, operators, and operands of the learning rule are all genomes to be evolved in this approach. Jordan et al. [2021] implemented Cartesian genetic programming (CGP) to evolve synaptic plasticity rules for learning reward-driven, error-driven, and correlation tasks. Confavreux et al. [2020] successfully applied covariance matrix adaptation evolution strategy to rediscover Oja's rule and an anti-Hebbian rule. Compared to the optimization algorithms discussed in the previous paragraphs, these approaches evolve the entire expression of the learning rule. They have a larger search space than methods that only evolve coefficients. Therefore, they are also more robust and flexible. Furthermore, the result of the evolution is a mathematical expression of the learning rule, which makes interpretation of learning behaviour easier. For these reasons, we will follow a similar approach in this paper.

3 Methodology

3.1 Neuron Model

There are a number of neuron models with varying levels of complexity and fidelity [Hodgkin and Huxley, 1952; Brunel and van Rossum, 1907]. Because the evolution of learning rules for training SNNs is computationally very expensive, a neuron model with low complexity is favorable. Furthermore, high fidelity regarding the biological neural system is not an essential factor for succesful evolution. For these reasons, the leaky integrate-and-fire (LIF) model is sufficient for the spiking neuron simulation here. Analogous to an RC circuit, the neuron membrane can be considered as a capacitor and the membrane potential U varies with the current I flowing into it, which is the summation process. When the membrane potential reaches the spiking threshold, the neuron releases a spike S and the membrane potential drops to the equilibrium or resting value. The leaky behavior describes the decay of the potential due to ion diffusion. The LIF model can be expressed mathematically as follows:

$$U_i^k = (1 - S_i^{k-1})\alpha U_i^{k-1} + (1 - \alpha) I_i^k \tag{1}$$

$$I_i^k = \sum_j w_{ij} S_j^k \tag{2}$$

where i, j indicate the post- and presynaptic neurons respectively, and k indicates the simulation timestep. The constant α is the membrane decay and w is the synaptic weight.

3.2 Reward Modulated Spike Timing Dependent Plasticity (R-STDP)

STDP is a learning rule fitted to experimental data of biological neurons [Markram *et al.*, 1997], and variants of STDP have been widely applied for training SNNs. The biological foundation is Hebb's theory, which is summarized as "cells that fire together wire together" [Hebb, 1949]. It was later extended with the long-term potentiation and long-term depression phenomena discovered in the rabbit hippocampus [Lømo, 1966]. The LTP phenomenon shows that a long-term increase in synaptic strength occurs when the postsynaptic neuron fires after the presynaptic spike in a certain time window. Oppositely, a long-term decrease takes place when the postsynaptic neuron fires before the presynaptic spike in a certain time window. Thus, in the basic STDP learning rule, synapse strength change is a function of the time difference between the pre- and postsynaptic spikes. In some variations of the basic STDP, the eligibility trace of the pre- and postsynaptic neurons or synapses are used to represent the time difference of spikes. To further imitate the reward driven learning behaviour in nature, reward-modulated STDP (R-STDP) introduces a reward signal in the

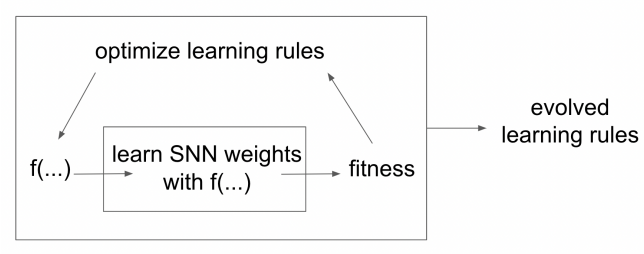


Figure 1: **Two learning loops of the evolving-to-learn algorithm.** The outer loop evolves the learning rule $f(...)$ to learn the task in the inner loop. The output of the inner loop is the fitness of the learning rule, which is used to select better learning rules.

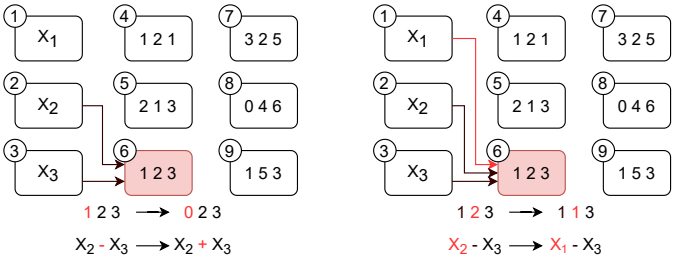


Figure 2: **An example of the mutation in CGP.** The left graph shows the mutation in the operator. The first index 1 is replaced by 0, the minus operator is mutated to a plus sign. The right graph shows the mutation in the operand. The index corresponding to the input x_2 is mutated to the index of the input x_1.

learning rule, making it suitable for training SNNs to perform reinforcement learning tasks. An example is the MSTDPET rule proposed by Florian [2017]:

$$w_{ij}(t+\Delta t) = w_{ij}(t) + \gamma R(t+\Delta t) E_{ij}(t+\Delta t) \quad (3)$$

$$E_{ij}(t+\Delta t) = E_{ij}(t) e^{-\Delta t/\tau_z} + \xi_{ij}(t) \quad (4)$$

$$\xi_{ij}(t) = X_j^+(t) S_i(t) + X_i^-(t) S_j(t) \quad (5)$$

$$X_j^+(t) = X_j^+(t-\Delta t) e^{-\Delta t/\tau_+} + A_+ S_j(t) \quad (6)$$

$$X_i^-(t) = X_i^-(t-\Delta t) e^{-\Delta t/\tau_-} + A_- S_i(t) \quad (7)$$

where γ is the learning rate and Δt is the simulation timestep. The reward R is updated at each timestep. The eligibility trace at each synapse E_{ij} is a low-pass filter over synaptic activity ξ_{ij}, which has a pre-before-post $(+)$ and a post-before-pre $(-)$ term. All τs represent decay time constants. Constants $A_\pm$ scale the contributions of spikes S to the neuron activity traces $X^\pm$.

3.3 Evolving-to-learn

The evolving-to-learn algorithm proposed by Jordan et al. [2021] can be considered as an optimization-based meta-learning algorithm with two loops, as visualized in Figure 1. The outer loop is an evolutionary search for the mathematical expressions of learning rules, which will be performed with CGP [Miller, 2007]. The learning rules are represented by an indexed 2D Cartesian graph. The graph has a fixed number of input, output, and internal nodes. Each internal node has a string of three integers: the first integer indicates the operator and the other two integers indicate the index of nodes to be connected. The output of the internal nodes can also be used as input to other internal nodes.

At the beginning of the evolution, the indexed graphs are randomly initialized. After each generation, the μ best solutions will be selected as parents and an offspring of size λ will be generated from the parents through mutation. Mutation can take place in both the operators and the connection of the nodes. An example of CGP with three columns and three rows is shown in Figure 2.

Learning rules generated by CGP will then be used to train SNNs to perform a particular task in the inner loop. The performance of the inner loop is in turn the fitness of the outer loop for the evolution. Summarized, the algorithm can be described by the following steps:[1]

1. Define the task to be learned for the inner loop and the fitness function.
2. Design the SNN architecture and determine encoding-decoding methods.
3. Specify the input signals for the evolutionary search.
4. Perform the evolution to discover learning rules with high fitness.

The same procedure will be implemented in our experiments. However, different from the work of Jordan et al. [2021] which used only the input signals of baseline learning rules, we also included neuronal states such as spikes and neuron traces, in order to give the evolution more freedom. Hyperparameters such as the membrane decay and threshold of spike are not included in the evolution; they are pre-tuned with the baseline learning rule and remain fixed.

4 Experiments

4.1 XOR Classification

We first test the proposed method on the elementary XOR classification task. Each XOR input is encoded as a spike train of 500 timesteps. 50 spikes are randomly distributed within the interval, generating two distinct patterns for 0 and 1 respectively. At the beginning of each learning epoch, different sets of input patterns are generated as training and test samples. The SNN has three layers of LIF neurons: 2 input neurons, 20 hidden neurons, and a single output neuron. The classification depends on the total number of output spikes. The output will be 1 if the output neuron spikes more than the average output spikes of one learning cycle; else, the output will be 0.

As a baseline learning rule, we use the MSTDPET rule [Florian, 2017], as given in Equations 3-7. Different from the Urbanczik and Senn's rule used in [Jordan *et al.*, 2021], MSTDPET updates connection weights every timestep of the simulation. Therefore, the reward should also be assigned to the network at each timestep. If the correct output is 1, a reward of 1 will be assigned to the synapse

[1]Code will be made open-source upon publication.

for each output spike released. If the correct output is 0, the reward is -1 for each output spike released. In the absence of output spikes, the reward for both cases is 0.

As discussed before, the signals available to the baseline learning rule will be included in the evolution, which in this case are E_{ij} and R. To expand the search space and endow more freedom to the evolutionary search, neuronal states that play a role in biological synaptic plasticity are also made available to the evolution: pre- and postsynaptic spikes S_i and S_j, and pre- and postsynaptic neuron traces with two different decays e_{i1}, e_{i2}, e_{j1} and e_{j2}. The general form of the learning rule is then expressed as follows:

$$\Delta w_{ij} = \gamma f(E_{ij}, R, S_i, S_j, e_{i1}, e_{i2}, e_{j1}, e_{j2}) \qquad (8)$$

Since the weight initialization is performed randomly, there can be cases that the initialization is already near an optimal solution, resulting in good performance, and also cases where the initialization is unfavorable for learning. Thus, each individual performs three trials of the experiment in the evolutionary search. The fitness function is defined to be the average of the end test accuracy of the three trials.

The results of 10 evolutionary runs are shown in Figure 3(a-b). It shows that two of the learning cycles successfully evolved the MSTDPET learning rule within 100 generations. Besides, it can be noticed that the term ER appeared in all of the evolved learning rules with high fitness, indicating that it is critical for learning the XOR task. This can be explained as follows. With temporal encoding, the two input neurons will spike simultaneously for pattern 0. Synapses between a postsynaptic neuron and the two input neurons will have an identical E at each timestep. Receiving a negative reward for each output spike released, the synapse weights will also be decreased by the same amount. However, when the label of the input is 1, the two input spike trains will be different. If the synapse connection is strong, there is a larger possibility that the postsynaptic neuron will spike right after the presynaptic spike resulting, in a larger synapse trace. Oppositely, connections with smaller weights tend to have smaller synaptic traces. Thus, after a number of learning cycles, strong connections will be stronger and the weak get weaker, which is a pattern favorable to solving the XOR task.

Aside from rediscovering the baseline learning rule, a learning rule that achieved a better performance was discovered, which has the expression $E(S_j + R) + S_j$. In addition to the synapse trace and reward, the presynaptic spike is also included in the learning rule. Similar to the MSTDPET rule, the evolved rule learns the XOR task by increasing the strength difference between the synapses connected to the two input neurons. With the baseline learning rule, weights will only be updated when the postsynaptic neuron spikes. The evolved rule also updates weights when the presynaptic neuron spikes. For pattern 0, weights of synapses connected to both input neurons will increase identically when there is only a presynaptic spike and decrease when there is only an output spike. For pattern 1, both the presynaptic spike and output spike will increase the strength difference between the synapses connected to two input neurons, thereby speeding up the learning as shown in Figure 3(c). Figure 3(d) demonstrates that after 500 learning epochs with the evolved

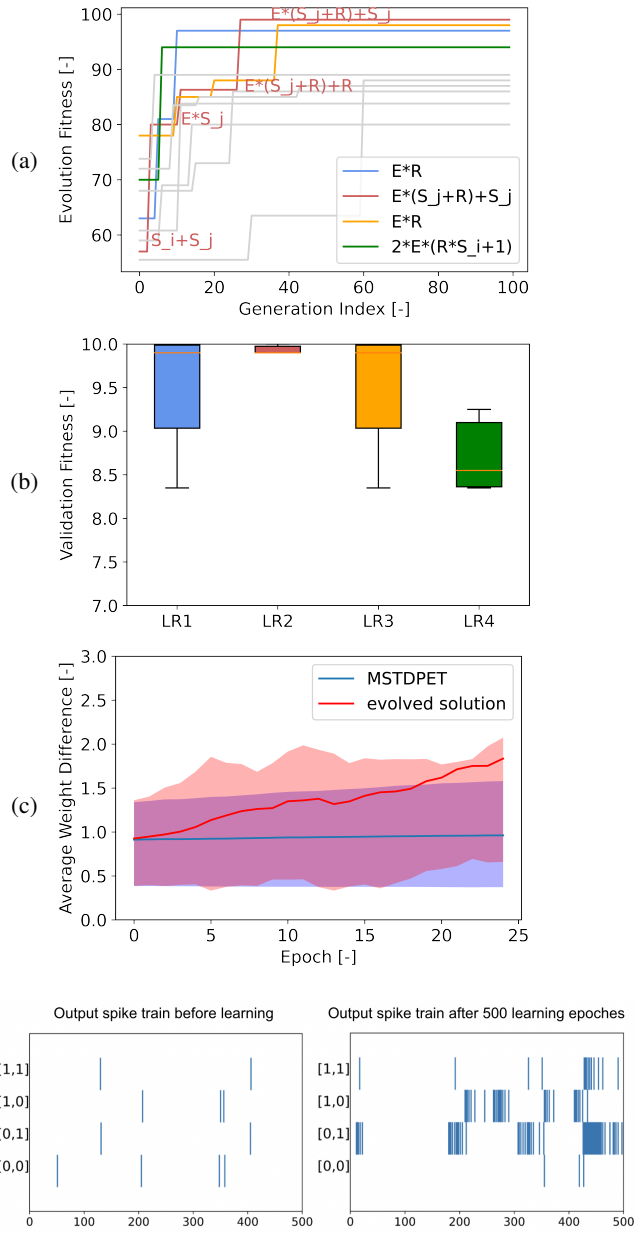


Figure 3: **Results of the evolution for learning the XOR task.** (a) The learning curve of the evolutionary search shows the fitness of the best solution at each generation. The grey curves are trials that failed to discover solutions with performance comparable to the baseline learning rule; the others are successful trials. The evolutionary process is also shown in the plot for the best learning trial. (b) The final evolved learning rules of the successful trials were validated with 50 random learning cycles. The average test accuracy at the end of the learning with the inter-quartile range (IQR) is plotted. (c) The plot shows the average variation of the difference between the weights connected to the two input neurons for both the baseline learning rule and the best evolved solution. The shaded area represents the IQR. (d) The output spike trains before and after learning with the evolved learning rule are visualized.

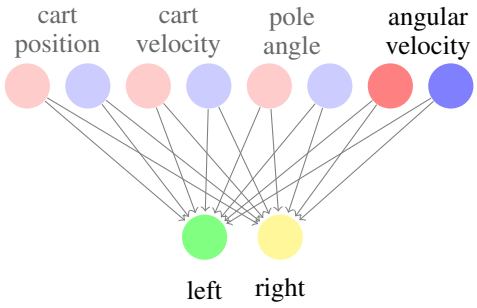


Figure 4: **The architecture of the SNN for the cart-pole task.** Each blue circle represents five neurons for negative values and each red circle represents five neurons for positive values. Only the input neurons corresponding to the angular velocity are used in the final experiment.

learning rule, the output spikes rate for input $[0, 1]$ and $[1, 0]$ is significantly larger.

4.2 Cart-pole Task

Next, the same evolving-to-learn approach was implemented to learn plasticity rules for training SNNs to perform the cart-pole task. The CartPole environment of OpenAI Gym[2] was used, where four observation states were available: cart position, cart velocity, pole angle, and pole angular velocity. The actions of the system are discrete: a force exerted on the cart towards the left or right. The SNN has two layers: an input layer of pairs of Poisson encoders and 2 output neurons. Each pair of Poisson encoders has 5 neurons for positive values and 5 neurons for negative values. The Poisson encoders convert the absolute value of the input state into a spike train with 50 network simulation timesteps. One of the binary encoders will be activated according to the sign of the state. An episode is a simulation cycle of the cart-pole with a number of environment timesteps. For each environment timestep, the model will receive a set of observation states. After being encoded into spike trains, the SNN will go through the network simulation. At the end of the network simulation, the system will take an action based on the output of the model. The 2 output neurons control the action of the cart. If the output neuron corresponding to the left action spikes more than the other neuron, a force towards the left will be applied to the cart. Otherwise, a force towards the right will be exerted. The architecture of the SNN is visualized in Figure 4.

After the system takes an action, the model will receive a reward from the environment. If the action is to the left and causes the pole to move to the center, the synapses connected to the left output neuron will receive a positive reward while the other synapses will receive an opposite reward. If the action to the left causes the pole to fall, the connection with the output neuron corresponding to the left will receive a negative reward, and the other connections will receive a positive reward.

The baseline learning rule for learning the cart-pole task is the multiplicative R-STDP rule used in [Shim and Li, 2017] to learn robot control tasks:

$$w_{ij}(t+\Delta t) = w_{ij}(t) + \gamma R(t+\Delta t) E_{ij}(t+\Delta t) w_{ij}(0) \quad (9)$$

Different from MSTDPET, this learning rule involves the initial weight at the beginning of the learning window $w_{ij}(0)$ to boost the performance. However, the problem with applying this learning rule is that the reward has to be available at each network simulation timestep while the SNN model will only receive the feedback after taking an action at the end of the network simulation. To solve this issue, the neuronal states at each network timestep were stored during the simulation. After the action is taken, the weight will be updated with the reward and stored states. Next, it was noticed that the baseline learning rule failed to perform the cart-pole task stably. This is because in cases where the pole falls and pushes the cart to move fast, the output gets dominated by a large-magnitude state, leading to very-high-rate input spike trains. For this reason we modified the experiment: 1) instead of receiving all four states (cart position, cart velocity, pole angle, and pole angular velocity), the network receives only the pole angular velocity (most essential information); 2) the network weights were only updated during a certain amount of timesteps at the beginning of each episode.

Similar to the XOR task, pre- and postsynaptic neuron activities and traces are also included in the evolution along with the input variables of the baseline learning rule. The synaptic plasticity rule can be expressed as follows:

$$\Delta w_{ij} = \gamma f(E_{ij}, R, f_i, f_j, e_{i1}, e_{i2}, e_{j1}, e_{j2}, w_{ij}(0)) \quad (10)$$

During the evolution, each candidate learning rule has three random trials of the task and each trial has 50 episodes. The maximum lifespan is 100 environment timesteps, and the episode will be terminated when the pole angle exceeds 15°. The fitness of the candidate learning rule is measured by the average time balance time of the last 5 episodes.

The results of 10 evolution runs are shown in Figure 5(a-b). It can be noticed that one of the evolution runs resulted in the baseline learning rule exactly and another run evolved a similar expression with comparable fitness: $Rw(0)(E - 1)$. For strong connections with large synaptic traces, the additional constant term is negligible and, therefore, has little influence on the learning performance. There are also two evolution runs that obtained high fitness during the evolution. However, they had an unstable learning performance and achieved a lower fitness in the validation. More importantly, the evolution also discovered a learning rule which obtained higher fitness and a more stable learning performance than other rules: $\Delta w_{ij}(t) = \gamma t R(t) E_{ij}(t) w_{ij}(0)^2$. Compared to the baseline learning rule, it includes all the inputs from the multiplicative R-STDP (Equation 9), but the initial weight has a power of two. It boosts the learning performance compared to the baseline learning rule as shown in Figure 5(c). When the strong connection results in the correct action and obtains a positive reward, the learning rule will further strengthen the connection pattern. However, when the strong connection results in a wrong action and obtains a negative reward, the learning rule will decrease the strong connection weights and increase the weak connection weights. The $w_{ij}(0)^2$ term will lead to the desired pattern faster. Figure 5(d) demonstrates

[2] https://gym.openai.com/envs/CartPole-v1/

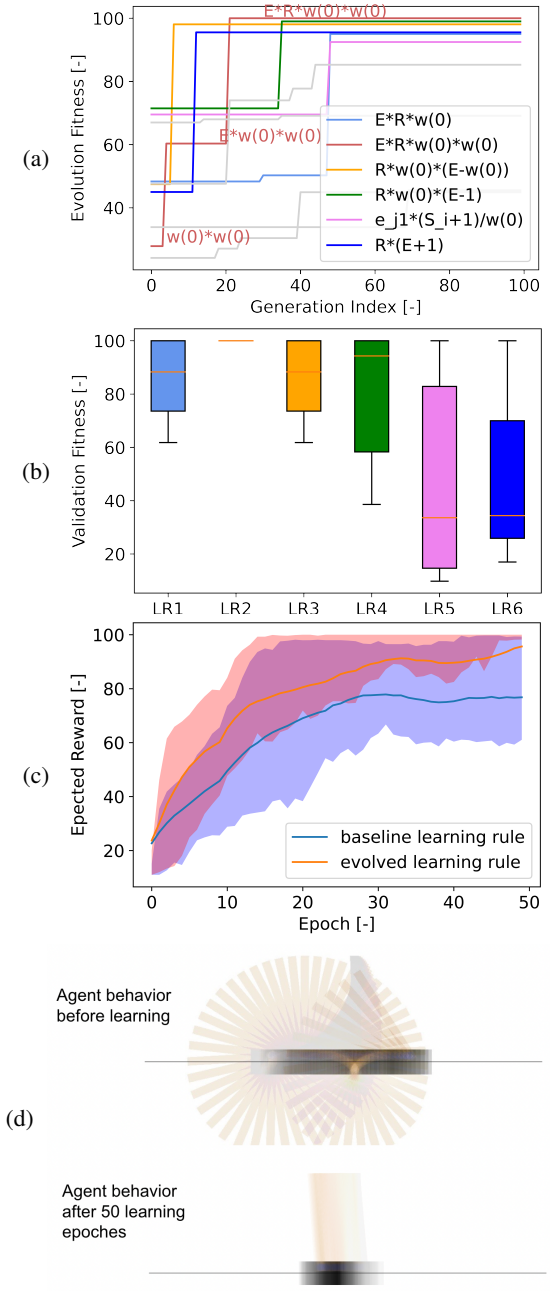


Figure 5: **Results of the evolution for learning the cart-pole task.** (a) The learning curve of the evolutionary search shows the fitness of the best solution at each generation. The grey curves are trials that failed to discover solutions with comparable performance as the baseline learning rule, the others are successful trials. The evolution process is also shown in the plot for the best learning trial. (b) The final evolved learning rules of the successful trials were validated with 50 random learning cycles. The average test accuracy at the end of the learning with IQR is plotted. (c) The plot shows the average learning curves of 50 validation runs for both the baseline learning rule and the evolved learning rule. The shaded area represents the IQR. (d) The agent behaviour trajectory of one episode before and after learning with the evolved learning rule are visualized. To show the entire trajectory, the episode was not terminated when the pole angle exceeded 15°.

that the pole remained balanced for the entire episode after 50 learning epochs with the evolved learning rule.

5 Discussion

In this article, we applied an evolving-to-learn algorithm to evolve mathematical expressions of synaptic plasticity learning rules using local input signals. The algorithm successfully learned learning rules for training SNNs on the XOR and cart-pole task. As demonstrated by the experiment results, both the XOR and cart-pole experiments (re)discovered the baseline learning rules and identified the essential terms for solving the tasks. This approach allows for better generalization of the synaptic plasticity rules to other similar tasks, since the learning rule can be customized based on these essential terms. Meanwhile, compared to previous work [Jordan *et al.*, 2021; Confavreux *et al.*, 2020], we expanded the search space and solved a more complex problem in the form of the cart-pole task, further proving the generality of the algorithm and the flexibility of the evolutionary approach. Additionally, for both experiments we discovered a new learning rule with higher performance and stability than the baseline solution.

During the experiments, the significance of the prior knowledge in the existing synaptic plasticity rules was revealed. Not only because the baseline learning rule provided a part of the input signals, but also because preliminary learning was necessary for tuning the hyperparameters of the network model to ensure that the model was active for solving the task. However, since the hyperparameters were fixed during the evolution, it may discourage the discovery of new learning rules which work with other sets of hyperparameters.

Moreover, there is a part of the experiment that is usually neglected by other works and other research on synaptic plasticity learning for SNNs. In this work, it was found that the design of the experiment has a large influence on the evolution of the learning rule. For instance, the encoding and decoding, the definition of the reward are all elements that are important to training the SNN model. If one of these elements is not properly defined, it can lead to low performance for both the evolution as well as the learning rules themselves. These design decisions are usually made through trial-and-error and iteration during the experiments. Therefore, although the evolving-to-learn algorithm reduces the workload for manually deriving and comparing different synaptic plasticity rules, there is still a large amount of work for designing the experiment properly.

In conclusion, the evolving-to-learn algorithm is an effective tool for helping to understand the learning procedure of an existing synaptic plasticity rule and assisting the manual generalization of the learning rule. It is also capable of discovering and optimizing learning rules with a predefined experimental setup. However, due to the dependence on prior knowledge and the sensitivity to the design decisions, the algorithm is not yet able to fully automatically discover new learning rules for a random new task. One next step towards a higher level meta-learning of synaptic plasticity rules can be including an automatic tuning of the hyperparameters in the evolution, which may allow the discovery of more new learning rules with the corresponding hyperparameters.

References

[Artola *et al.*, 1990] A. Artola, S. Brocher, and W. Singer. Different voltage-dependent thresholds for inducing long-term depression and long-term potentiation in slices of rat visual cortex. *Nature*, 347(6288):69, 1990.

[Bengio *et al.*, 1991] Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. Learning a synaptic learning rule. *IJCNN-91-Seattle International Joint Conference on Neural Networks*, ii:969 vol.2–, 1991.

[Bengio *et al.*, 2007] Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. On the optimization of a synaptic learning rule. 2007.

[Bohnstingl *et al.*, 2019] Thomas Bohnstingl, Franz Scherr, Christian Pehle, Karlheinz Meier, and Wolfgang Maass. Neuromorphic hardware learns to learn. *Frontiers in Neuroscience*, 13, 2019.

[Brunel and van Rossum, 1907] N. Brunel and M. van Rossum. From frogs to integrate-and-fire. *Biological Cybernetics*, 97(5):337–339, 1907.

[Confavreux *et al.*, 2020] Basile Confavreux, Friedemann Zenke, Everton J. Agnes, Timothy Lillicrap, and Tim P. Vogels. A meta-learning approach to (re)discover plasticity rules that carve a desired function into a neural network. *Advances in Neural Information Processing Systems*, pages 1–11, 2020.

[Davies *et al.*, 2018] Mike E. Davies, Narayan Srinivasa, and Tsung-Han Lin *et al.* Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38:82–99, 2018.

[DeBole *et al.*, 2019] Michael V. DeBole, Brian Taba, and Arnon Amir *et al.* Truenorth: Accelerating from zero to 64 million neurons in 10 years. *Computer*, 52:20–29, 2019.

[Florian, 2017] Răzvan V. Florian. Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. 19(6):1468–1502, 2017.

[Gerstner *et al.*, 2014] Wulfram Gerstner, W. M. Kistler, Richard Naud, and Liam Paninski. *Neuronal Dynamics*. Cambridge University Press, 2014.

[Hagenaars *et al.*, 2020] Jesse J. Hagenaars, F. Paredes-Vallés, Sander M. Bohté, and Guido de Croon. Evolved neuromorphic control for high speed divergence-based landings of mavs. *IEEE Robotics and Automation Letters*, 5:6239–6246, 2020.

[Hebb, 1949] D.O. Hebb. *The Organization of Behavior*. John Wiley, 1949.

[Hodgkin and Huxley, 1952] A.L. Hodgkin and A.F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerves. *The Journal of Physiology*, 117(4):500–544, 1952.

[Howard and Elfes, 2014] David Howard and Alberto Elfes. Evolving spiking networks for turbulence-tolerant quadrotor control. 2014.

[Jordan *et al.*, 2021] J Jordan, M Schmidt, W Senn, and M.A Petrovici. Evolving to learn: discovering interpretable plasticity rules for spiking networks. *arXiv preprint*, 2021.

[Lømo, 1966] T. Lømo. Frequency potentiation of excitatory synaptic activity in the dentate area of the hippocampal formation. *Acta Physiol. Scand.*, 68(277):128, 1966.

[Maas, 1997] Wofgang Maas. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 14:1659–1671, 1997.

[Markram *et al.*, 1997] H Markram, J Lübke, M Frotscher, and B Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps. *Science*, 275(5297):213–215, 1997.

[Miller, 2007] Julian Francis Miller. Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study. 2007.

[Neftci *et al.*, 2019] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks. *ArXiv*, abs/1901.09948, 2019.

[Niv *et al.*, 2002] Y. Niv, D. Joel, I. Meilijson, and E. Ruppin. Evolution of reinforcement learning in uncertain environments: A simple explanation for complex foraging behaviors. *Adaptive Behavior*, 10(1):5–24, 2002.

[Oja, 1982] E. Oja. Simplified neuron model as a principal component analyzer. *J. Math. Biology*, 15:267–273, 1982.

[Pfister and Gerstner, 2006] Jean-Pascal Pfister and Wulfram Gerstner. Triplets of spikes in a model of spike timing-dependent plasticity. *The Journal of Neuroscience*, 26:9673 – 9682, 2006.

[Risi and Stanley, 2010] S Risi and K Stanley. Indirectly encoding neural plasticity as a pattern of local rules. *International Conference on Simulation of Adaptive Behaviour*, 2010.

[Shim and Li, 2017] Myung Seok Shim and Peng Li. Biologically inspired reinforcement learning for mobile robot collision avoidance. *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 3098–3105, 2017.

[Shrestha and Orchard, 2018] S. Shrestha and G. Orchard. Slayer: Spike layer error reassignment in time. In *NeurIPS*, 2018.

[Stanley *et al.*, 2003] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. Evolving adaptive neural networks with and without adaptive synapses. In *Proceedings of the 2003 Congress on Evolutionary Computation*, Piscataway, NJ, 2003. IEEE.

[Stone, 2016] James V. Stone. Principles of neural information theory a tutorial introduction. 2016.

[Toyoizumi *et al.*, 2005] Taro Toyoizumi, Jean-Pascal Pfister, Kazuyuki Aihara, and Wulfram Gerstner. Generalized bienenstock-cooper-munro rule for spiking neurons that maximizes information transmission. *Proceedings of the National Academy of Sciences of the United States of America*, 102 14:5239–44, 2005.

A Hyperparameters

Table 1 lists the hyperparameters used in the SNNs and evolution for XOR and cart-pole tasks respectively.

Hyperparameters	**XOR**	**Cart-pole**
number of spikes per pattern (N)	50	-
number of patterns per training epoch (M)	4	-
number of episode per epoch (N_{ep})	-	100
simulation duration (T)	500	50
environment duration (T_{env})	-	100
number of training epoches (N_e)	500	50
SNN architecture	[2, 20, 1]	[10, 2]
membrane potential decay (τ_V)	10	10
synapse trace decay (τ_e)	100	100
spike threshold (θ)	100	1
activity decay (τ_+, τ_-)	10	2
pre- and postspike constants (A_+, A_-)	+1, -1	+1.5, -0.5
reward (R)	+1, -1	+1, -0.5
leanring rate (η)	5 e-3	e-5
number of parents (μ)	20	50
number of offsprings (λ)	20	50
number of generations (N_g)	100	100
CGP architecture	2×12	2×12
operators	+ - $\times$ \constantfloats	+ - $\times$ \constantfloats
mutation rate	0.3	0.3

Table 1: Hyperparameters for the XOR and cart-pole experiments.

II

Literature Study

2

Spiking Neural Networks

Along with the development of artificial intelligence, artificial neural networks have also been evolving throughout the decades. According to Maass (1997), neural networks can be classified into three generations. Starting from the computation units based on the McCulloch-Pitts models, which are also referred to as perceptions and threshold gates, various neural networks were generated. Although the neural networks in this stage only generate digital outputs, they already had the ability to model Boolean functions. What is now commonly referred to as ANN is the neural network of the second generation. The activation functions started playing an important role in the neural networks, which allows the networks to generate continuous outputs and learn more efficiently. ANNs in the second generation such as the feed-forward neural networks have been rapidly developed into a powerful tool for solving artificial intelligence tasks in various fields. However, the learning algorithms of ANNs of the second generation are usually based on the gradient descent method such as backpropagation, which are efficient for learning but not biologically plausible (Bengio et al., 2015). In order to improve the biological plausibility of the learning algorithms of ANNs and further mimicking the natural neural systems, research in spiking neural networks (SNNs) has risen in recent decades. SNNs are known as third-generation neural networks. Compared to ANNs, SNNs are more bio-mimetic. SNNs are composed of biological neurons which emit spikes. An introduction to the SNNs will be presented in this chapter starting from a brief discussion of its background in neuroscience in section 2.1. In section 2.2, several commonly used neuron models will be introduced. It is followed by an introduction to the typical learning algorithms for SNNs in section 2.3.

2.1 Neuroscience Background

The nervous system in nature is a highly complex system that transmits signals and coordinates actions towards changes in the environment. Because of its significant functions and importance in biological systems, there are various interests in neuroscience that study the nervous system from different aspects. For the field of bio-inspired artificial intelligence, the understanding of the neurons and the signal transmission plays an important role. Thus, the following paragraphs will introduce the nervous system from these aspects based on the book of Gerstner et al. (2014).

Neurons are the elementary processing units in biological nervous systems. Ideal neurons consist of three components: dendrites, soma, and axon as distinguished in the sketch in Figure 2.1. These components analogize the input device, the central processing unit, and the output device in a computer. Dendrites collect input signals from other neurons and transmit them to the soma. The input signal will then accumulate in the soma. Once the total accumulated signal exceeds a certain threshold, the axon will generate an output signal and deliver it to other neurons. Neurons are connected by synapses. For a pair of connected neurons, the neuron which delivers the output signal is called the presynaptic neuron and the neuron which receives the signal is called the postsynaptic neuron. The signals transmitted between neurons are in the form of electrical pulses, which are also known as action potentials or spikes. The signal sequence generated by one neuron is the spike train. One single spike does not

contain information, but it is the pattern of the spike train that encodes the information for transmission.

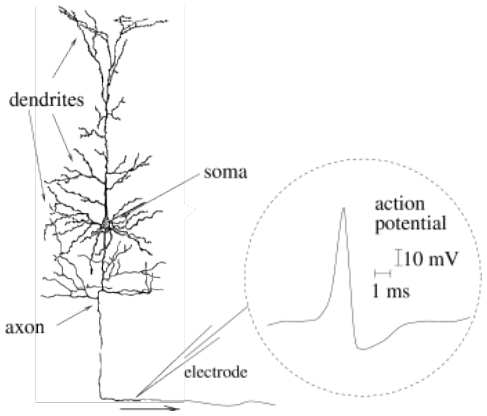


Figure 2.1: Sketch of a single neuron by Ramón y Cajal adapted from Gerstner et al. (2014).

The signal transmission between neurons relies on the potential difference between the interior of the neuron and the surroundings. When there is no input signal, the neuron is at a constant resting membrane potential. The membrane potential changes at the arrival of input signals and decays with time until reaching the resting membrane potential again. Synapses can be categorized into two different types based on the sign of the potential variations: excitatory synapses lead to positive changes in potential while inhibitory synapses result in negative changes.

The scheme of the potential variation is different between neurons in different nervous systems. In the case of a nervous system with chemical synapses, the axon of the presynaptic neuron releases chemical substances when its membrane potential is above the firing threshold. The chemical substances are known as neurotransmitters. The receptor in the dendrite of the postsynaptic neuron will detect the neurotransmitters and thereafter open specific channels of corresponding ions. The ions flow into the neuron cell and cause changes in the membrane potential. A sketch of the variation of the membrane potential of an excitatory synapse with spikes received from the presynaptic neuron is shown in Figure 2.2. The green pulses indicate the incoming spike train. The blue curve demonstrates that the membrane potential rises when the spike arrives and decays with time. When the accumulated membrane potential reaches the firing threshold V, the neuron will release a spike and the membrane potential will drop below the rest potential u_{rest}. The neuron will then enter the refractory phase when the neuron is inactive to input spikes until the membrane potential rises to the rest level again.

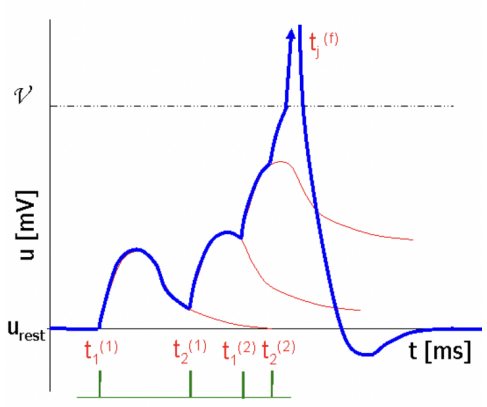


Figure 2.2: Variations of action potential adapted from Paugam-Moisy (2012).

2.2 Neuron Models

Based on the discoveries and experimental data in the field of neuroscience as mentioned before, the neurons can be expressed by a mathematical model to mimic the membrane potential variations of neurons in nature. The mathematical model is therefore called the neuron model. The neuron models are not only applied in research in neuroscience but also implemented as the unit components of SNNs. Neuron models in SNNs will respond to input signals and transmit signals like biological neurons. There are various neuron models with different fidelity and complexity, which can be selected for tasks with different requirements in accuracy and computational cost. This section will introduce three commonly used neuron models in the field of SNNs: the Hodgkin-Huxley model, the integrate-and-fire model, and the Izhikevich model.

2.2.1 Hodgkin-Huxley (HH) model

The Hodgkin-Huxley (HH) model developed by Hodgkin and Huxley (1952) is considered the most accurate mathematical neuron model. It includes all the elements of the neuron into the model and treats them as electrical components. The model treats the lipid bilayer as a capacitance C_m. The electrochemical gradients are considered as voltage sources E for each type of ion flux. In the original HH model, the neuron membrane has three ion channels: the leakage channel, Sodium channel, and Potassium channel. Hodgkin and Huxley discovered that the conductance of the ion channels is a function depending on the voltage and time (voltage-gated). Thus, each voltage-gated ion channel can be represented by a nonlinear conductance g and the value of the conductance varies with the open or close of the channel. The main achievement of the HH model is it successfully describes the conductance of each ion channel as a function of time and voltage and introduces "gating" variables m, n, h, which represent the possibility of the open or close of the channels respect to time. The conductance of the leaky channel is a constant value g_L. Along with the current source I, the mathematical expression of the HH model can be represented by the following set of four ordinary differential equations:

$$C_m \frac{dv}{dt} = -g_L(V - E_L) - g_{Na} m^3 h(V - E_{Na}) - g_K n^4 (V - E_K) + I \tag{2.1}$$

$$\frac{dn}{dt} = \alpha_n(V)(1 - n) - \beta_n(V)n \tag{2.2}$$

$$\frac{dm}{dt} = \alpha_m(V)(1 - m) - \beta_m(V)m \tag{2.3}$$

$$\frac{dh}{dt} = \alpha_h(V)(1 - h) - \beta_h(V)h \tag{2.4}$$

where α and β are rate constants that depend on voltage.

The HH model can describe complex biological processes. However, it is seldom used in large-scale simulations due to its complexity. To reduce the computational cost, there are also some simplified representations of the HH model such as the BVP model proposed by FitzHugh (1961), which ignores several neuron behaviors.

2.2.2 Integrate-and-Fire (I&F) model

The Integrate-and-fire model (I&F) proposed by Brunel N. (1907) is the most widely used neuron as a first and rough approximation of the neuron to prove analytical results and understanding the basic properties of large networks of neurons. Compared to the HH model, I&F is much simpler since it only describes two key mechanisms of the neuron dynamics: the summation process and the trigger of action potentials. The summation process is the potential variation due to the incoming current. In analog to an RC circuit, the neuron membrane can be considered as a capacitor with a capacitance of C_m, and the membrane potential increases with the current flowing into it. When the potential reaches the spiking threshold, the neuron releases a spike and the membrane potential drops to the equilibrium value E_m, which describes the trigger of action potentials.

There are various variations of the I&F model, a commonly used model is the leaky integrate-and-fire (LIF) model. The LIF model also includes the leaky behavior of the membrane, which allows the current to flow across the membrane with a resistance of R_m. The mathematical expression of the LIF model is shown below:

$$C_m \frac{dV}{dt} = -\frac{V - E_m}{R_m} + I \tag{2.5}$$

Another variation of the I&F model is the adaptive integrate-and-fire model, which involves the leaky integration voltage and adaptation variables. By including adaptation variables, the model describes the neuronal adaptation referring to the variation of the firing threshold with time. An advanced model is the fractional-order LIF model. Instead of using several adaptation variables, the fractional-order LIF model uses a single adaptation variable to describe the variation of membrane potential with time (Gerstner et al., 2014).

2.2.3 Izhikevich Model

The Izhikevich model (Izhikevich, 2003) is a neuron model that keeps a good balance between fidelity and computational cost. It is also inspired by the Hodgkin-Huxley model but only involves four parameters and two differential equations. Thus, the Izhikevich model is much less computationally expensive than the HH model. In the meantime, it is relatively more biologically accurate than the I&F model. Apart from the basic mechanisms of the neuron dynamics as described by the integrate-and-fire model, it is also capable of describing the bursting behavior of cortical neurons. The mathematical expressions of the model are two ordinary differential equations shown in Equation 2.6 and Equation 2.7.

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \tag{2.6}$$

$$\frac{du}{dt} = a(bv - u) \quad if \quad v \geq 30mV, \quad then \begin{cases} v = c \\ u = u + d \end{cases} \tag{2.7}$$

Variables v and u are membrane potential and membrane recovery variables respectively. The parameters a,b,c,d are dimensionless constants that describe the characteristics of the recovery variable and membrane potential.

2.3 Learning in Spiking Neural Networks

Although SNNs are favorable for the low computational cost, they are not yet widely used in practical applications. One critical problem is the difficulty in training SNNs effectively. Nowadays, the learning algorithms in the field of SNNs can be categorized into four approaches: unsupervised learning, supervised learning, reinforcement learning, and neuroevolution. The principle of each learning approach will be introduced in this section along with several examples.

2.3.1 Unsupervised Learning

Similar to unsupervised learning in ANNs, unsupervised learning in SNNs is also intended for learning tasks without labels. The main difference is that unsupervised learning in SNNs is usually based on the synaptic plasticity mechanism in biological neural systems. The first explanation of synaptic plasticity is the theory proposed by Hebb (1949), which is also known as Hebb's rule. Hebb proposed that the synaptic connection between two adjacent neurons enhances if one neuron fires immediately after the other repeatedly, and reduces otherwise. It is also summarized as "cells that fire together wire together". Hebb's rule was later proved by the long-term potentiation (LTP) and long-term depression (LTD) phenomena discovered in the rabbit hippocampus (Lømo, 1966). It was found that a long-term increase in synaptic strength occurs when the postsynaptic neuron fires after the presynaptic neuron in a certain time window, which is named the LTP phenomena. Oppositely, a long-term decrease takes place when the postsynaptic neuron fires before the presynaptic neuron in a certain time window. Later in 1996, Markram demonstrated the spike-timing-dependent plasticity (STDP) rule based on

the qualitative expression of the LTP phenomena and LTD phenomena discovered with dual patch clamping techniques (Markram et al., 1997). Compared to Hebb's rule, STDP emphasizes the effects of the temporal order of the neuron pairs. The mathematical expression of the STDP rule was later established by Song et al. (2000) as shown in Equation 2.8.

$$F(\Delta t) = \begin{cases} A_+ \exp(\Delta t/\tau_+) & if \quad \Delta t < 0 \\ \\ A_- \exp(-\Delta t/\tau_-) & if \quad \Delta t > 0 \end{cases} \tag{2.8}$$

The synaptic strength is represented by the peak conductance of the synapse. The equation describes the relationship between the synaptic strength and the time difference between the pre- and postsynaptic spikes Δt. The time difference is computed as $\Delta t = t_j - t_i$, where t_j is the time of the presynaptic spike and t_i is the time of the postsynaptic spike. When Δt is negative, the presynaptic neuron spikes before the postsynaptic neuron resulting in the LTP phenomena. Thus, the amplitude A_+ is positive. Oppositely, the amplitude for the LTD phenomena A_- is negative. The constants τ_+ and τ_- are positive parameters describing the time window of the LTP and LTD phenomena respectively. In other words, they determine the time interval between the presynaptic and postsynaptic spikes over which distinct synapse variation occurs. In the experiments of Song et al. (2000), they found a typical time window of $20ms$ for both τ_+ and τ_-. Below is a visualization of the model proposed by Song et al. (2000) as shown in Figure 2.3. The figure shows rough equality between the LTP and LTD phenomena. However, there are experimental discoveries demonstrating that synaptic weakening has a larger effect than synaptic strengthening (Gerstner et al., 1997). It can also be realized by varying the amplitude and time window of the model.

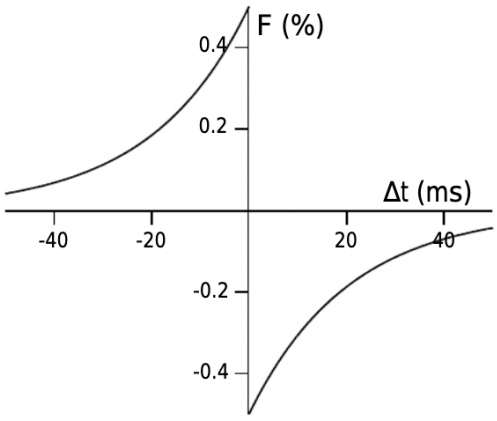


Figure 2.3: Visualization of the STDP mathematical model proposed by Song et al. (2000)

Later experiments also found that the original STDP model with pairs of presynaptic spikes and postsynaptic spikes can not fully describe the communication scheme for all types of neurons (Morrison et al., 2007). Morrison et al. (2008) developed a symmetric triplet model for the cortical neurons. STDP of a triplet model depends on three connected neurons: pre-post-pre or post-pre-post. A variation of the triplet model was proposed by Pfister et al. (2006). Aside from neuron and synapse parameters in the original STDP rule, they also involved traces of the pre and postsynaptic neurons in the model.

Theories in synaptic plasticity have been adapted and implemented in training SNNs with unsupervised learning algorithms in recent years. Diehl and Cook (2015) applied both the original pair-based STDP and triplet STDP on training a two-layer SNN on the MNIST task. They also implemented the concept of eligibility trace at each synapse in their learning rule and updated it at each timestep. The network trained with the triplet STDP achieved a test accuracy of 95% in the end. The performance of the one trained with the pair-based STDP is slightly lower, but also obtained an accuracy of around 93%. Guo et al. (2019) implemented a variation of the standard pair-based STDP model, where a softbound was applied to the synapse strength (Kistler and Hemmen, 2019). The STDP characteristics were designed

according to the resistive random access memory (RRAM) devices. By training the SNNs on MNIST successfully, they revealed the feasibility of performing unsupervised learning in RRAM array-based neuromorphic systems.

2.3.2 Supervised Learning

Supervised Learning is referred to train neural networks by mapping input to target output on tasks with labeled training data. One of the most popular learning algorithms for supervised learning is backpropagation. It is a gradient-based method that updates connection weights by computing the gradient of the loss with respect to weights for each pair of input and output in the training dataset. The loss is a function that depends on the difference between the output and the label, which is usually cross-entropy or minimum square loss. Backpropagation is widely used in training ANNs since it is fast, simple, flexible, and general. Attempts have also been made to train SNNs with backpropagation. However, the discrete nature of SNNs makes it difficult for computing gradients.

There are several solutions that deal with discreteness. They are categorized into two approaches by Neftci et al. (2019). The first approach smooths the SNN model to obtain a well-behaved gradient for direct optimization, which can be realized by various methods such as softening the nonlinearity of the neuron model or applying a rate-based coding scheme. A widely used solution of the first category is the SpikeProp algorithm proposed by Bohte et al. (2002), which is also one of the earliest backpropagation algorithms for training SNNs. In their method, all the neurons are only allowed to spike once. The gradients of the hidden layers are computed by linearizing the analytical expressions of the fire time. The loss is defined as a function depending on the time difference of the output spikes and target spikes. They implemented the SpikeProp in a multi-layer feed-forward SNN and successfully learned the XOR classification task. The SpikeProp algorithm was further improved and expanded by including the neuron decay and spike threshold to weight update and applying adaptive learning rate (Schrauwen and Van Campenhout, 2004; Fang et al., 2012). The enhanced SpikeProp algorithm has faster convergence and can be applied in non-linear neuron models. However, the algorithm based on SpikeProp can not be used to train complex tasks due to the limitation that neurons are only allowed to spike once. A breakthrough is an algorithm proposed by Booij and Nguyen (2005), they only limited the neurons in the output layer to spike once while neurons in the input and hidden layers had no limitation in spikes. A similar approach is a Multi-SpikeProp algorithm raised by Ghosh-Dastidar and Adeli (2009). Both algorithms showed better performances in more complex tasks than the original SpikeProp. The second approach implements surrogate gradients instead of changing the model. Surrogate gradients can adjust the locality of the optimization algorithm or replacing the derivative of the spiking nonlinearity with that of a smooth function (Neftci et al., 2019).

Apart from training with backpropagation, synaptic plasticity can also be implemented in supervised learning of SNNs by introducing teaching signals. The first example that applies the Hebbian rule in supervised learning of SNNs is the research done by Ruf and Schmitt (1997). They used the time of the target spikes as the teaching signal, and the output neuron learned to spike at the targeted time by updating the synaptic connection with the time difference between the output spike and the target spike. Another popular supervised learning algorithm for SNNs is the remote supervised method (ReSuMe), which applied the STDP rule on the weight update as shown in Equation 2.9:

$$\Delta W = (S_d(t) - S_i(t))[a_d + \int_0^\infty a_{di}(s) S_j(t-s) ds] \tag{2.9}$$

where $S_d(t), S_i(t), S_j(t)$ are target spike train, postsynaptic spike train, and presynaptic spike train respectively. The sign of a_d indicates the type of synapse, positive is for excitatory synapse and negative for inhibitory synapse. The function a_{di} describes the learning window of the STDP rule.

The ReSuMe has been successfully applied on training single-hidden-layer SNNs in various temporal or spatial pattern classification and recognition tasks. The algorithm was further expanded to multi-hidden-layer SNNs by applying the SpikeProp method, which allows the learning of complex pattern classification tasks.

In addition to backpropagation and synaptic plasticity approaches, another widely used method for training SNNs with supervised learning is converting ANNs to SNNs. This approach uses a kernel function to convert the spike trains to convolutionary signals, then training the networks as the supervised learning of ANNs. An example is the Spike Pattern Association Neuron (SPAN) algorithm proposed by Mohemmed and Schliebs (2012). They used the kernel function $\alpha(t) = e\tau^{-1}te^{-t/\tau}H(t)$ to convert the spike train signals, where $H(t)$ is the Heaviside function. After the signals are converted to convolutionary functions, the weights can be updated by the Widrow-Hoff rule.

2.3.3 Reinforcement Learning

Reinforcement learning is a bio-inspired learning mechanism that trains the intelligent agent to take actions based on observations of the environment to maximize the reward (Hu et al., 2020). The basic model of reinforcement learning can be described by the Markov decision process (MDP) (van Otterlo and Wiering, 2012). The model consists of four elements: the state space S, the action space A, p representing the probability of a certain transition of state under the action, and r is the reward of an action that results in a certain transition of states. Figure 2.4 demonstrates an example of an MDP, where the state space consists of three states and the action space consists of two actions. The values besides the black arrows indicate the possibility of transition between two states under the action a. The goal of reinforcement learning is to find an action policy that optimizes the total reward obtained from the environment in a certain duration. A solution to this problem has been studied intensively in the past decades and various algorithms have been proposed. Typical algorithms are the Monte-Carlo methods, temporal difference methods, and policy search methods (Yuxi, 2017). Due to the robustness and the ability in solving complex tasks, reinforcement learning has become a popular learning rule aside from supervised learning and unsupervised learning and is widely used in various fields such as control theory and swarm intelligence (Hu et al., 2020; Hüttenrauch et al., 2019).

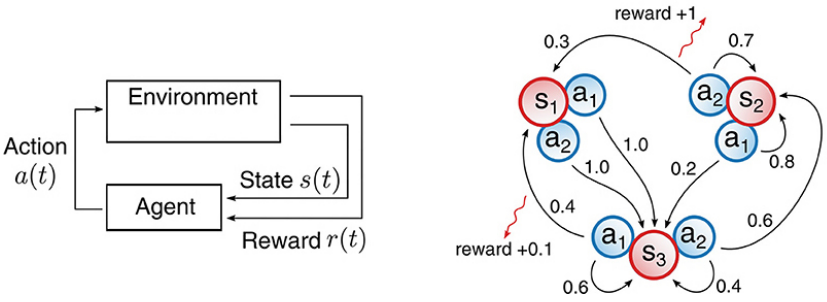


Figure 2.4: Visualization of MDP adapted from Bohnstingl et al. (2019)

Compared to other learning algorithms, reinforcement learning is more similar to the learning behaviour in biological systems that adjust actions based on the environment. Thus, it is natural to associate reinforcement learning with biologically plausible neural networks such as SNNs. Before reinforcement learning was applied in SNNs, the hypothesis of combining reward in synaptic plasticity was supported by research in ANNs with binary elements or threshold gates (Barto, 1985; Alstrøm and Stassinopoulos, 1995). The early algorithms which modulate the synaptic plasticity for SNNs are proposed by Seung (2003) and Xie and Seung (2004). The former method considers the action as the release of a neurotransmitter vesicle instead of a spike and the goal is to optimize the parameter which controls the release of the vesicle. The latter realized reinforcement learning by correlating fluctuations with a reward signal. One of the early examples that implemented SNNs trained with reward modulated synaptic plasticity rule was the research done by Soula et al. (2005). They developed an SNN controller for obstacle avoidance by applying anti-Hebbian STDP when the robot hits an obstacle and STDP opposite.

The definite model of reward-modulated STDP (RSTDP) was later proposed by Florian (2007). Apart from the reward signal, Florian (2007) also investigates the influence of eligibility trace. Two approaches were proposed, the approach that involves the synapse trace is named MSTDPET and the other without the trace is named MSTDP. The learning rules for both approaches are shown in Equation 2.10 and

Equation 2.11 respectively. The constant γ is the learning rate and δt is the timestep. The reward r is updated at each timestep.

$$w_{ij}(t+\delta t) = w_{ij}(t) + \gamma \delta t r(t+\delta t) z_{ij}(t+\delta t) \tag{2.10}$$

$$w_{ij}(t+\delta t) = w_{ij}(t) + \gamma r(t+\delta t) \xi_{ij}(t) \tag{2.11}$$

The eligibility trace at each synapse is computed by Equation 2.12.

$$z_{ij}(t+\delta t) = z_{ij}(t) + \xi_{ij}(t) \exp(-\delta t/\tau_z) \tag{2.12}$$

The dynamics of ξ_{ij} describes the presynaptic trace P_{ij}^{+} and postsynaptic trace P_{ij}^{-}, which are updated by presynaptic spike f_j and postsynaptic spike f_i. The amplitude A_+ is positive and A_- is negative describing the LTP and LTD phenomena respectively. The constants τ_z, τ_+, τ_- are decay parameters.

$$\xi_{ij}(t) = P_{ij}^{+}(t) f_i(t) + P_{ij}^{-}(t) f_j(t) \tag{2.13}$$

$$P_{ij}^{+}(t) = P_{ij}^{+}(t-\delta t) \exp(-\delta t/\tau_+) + A_+ f_j(t) \tag{2.14}$$

$$P_{ij}^{-}(t) = P_{ij}^{-}(t-\delta t) \exp(-\delta t/\tau_-) + A_- f_i(t) \tag{2.15}$$

In the research of Florian (2007), both the MSTDP and MSTDPET approaches successfully learned the rate-encoded and temporal-encoded XOR task, while the MSTDPET learns faster for both experiments and achieved better performance for the temporal-encoded experiment. Based on these models, several applications of training SNN controllers on reinforcement learning tasks have been performed. Bing et al. (2018) applied the RSTDP rule to train an SNN controller to perform lane-keeping tasks. Shim and Li (2017) modified the standard RSTDP rule to a multiplicative RSTDP and trained the SNN controller to avoid collisions.

Apart from modulating the STDP rule with reward, there are also approaches that apply temporal difference modulated STDP (TD-STDP). Frémaux et al. (2013) applied the continuous TD learning to an actor-critic SNN and trained the controller to perform continuous maze and cart-pole tasks. Chung and Kozma (2020) further extended the TD-STDP to training SNNs to perform discrete tasks by introducing feedback modulation.

2.3.4 Evolutionary Algorithms

Evolutionary algorithm (EA) is a general term referring to optimization algorithms that mimic the evolution process in nature (Vikhar, 2016). A pseudo-code demonstrating its general scheme is shown in Figure 2.5. Basic EA consists of four steps: generating the initial population, evaluating each individual by the fitness function, selecting individuals with high fitness, and breeding the next generation. The goal is to evolve individuals with high fitness at the end of a certain number of generations. EA is a broad field that contains various techniques with different implementation details. In the field of ANN, EA is usually known as neuroevolution. Aside from evolving the connection weights of the neural networks, neuroevolution algorithms can also be applied to optimize the hyper-parameters, the topology of the network, and the learning rules (Risi and Togelius, 2017).

Similar to neuroevolution in ANNs, various evolutionary algorithms have been proposed for evolving different components of the SNNs. For example, the research of Pérez et al. (2018) evolved the connection weights and the type of neuron for an SNN controller to perform dynamic system control. Hagenaars et al. (2020) evolved the connection weights and neuron characteristics for performing divergence-based landing tasks with a real-world micro air vehicle. Another widely used neuroevolution approach in SNNs is based on the neuroevolution of augmenting topologies (NEAT) algorithm developed by Stanley and Miikkulainen (2002). The NEAT algorithm does not only evolve the connection weights but also the topology of the networks. Qiu et al. (2018) evolved the SNN controller with the NEAT algorithm to perform the cart-pole task. Another algorithm that learns the network topology is the evolving spiking neural network (eSNN) proposed by Lobo et al. (2018). The architecture of the eSNN consists of three

Algorithm 1 General Scheme of Evolutionary Algorithms

```
INITIALISE a population of candidate solutions
EVALUATE each individual & ASSIGN fitness values
while (terminate condition not met) do
    SELECT parents
    RECOMBINATION parents
    MUTATE offspring
    EVALUATE new candidates
    SELECT individuals for the next generation
end while
```

Figure 2.5: Pseudo code of EAs adapted from Qiu et al. (2018)

layers: input data, encoding, and output layer. The evolution is only involved in the output layer to update the connection weights.

Aside from evolving the weights and hyperparameters of SNNs, evolutionary algorithms have also been developed to optimize the learning rules of SNNs. Niv et al. (2002) and Toenlli and Mouret (2011) applied evolutionary algorithms to optimize the coefficients of the Hebbian learning rule. The adaptive HyperNeat algorithm proposed by Risi and Stanley (2010) is an extended version of NEAT which evolves a neural network representing the weight update function. Since these algorithms do not only evolve the weights of the SNNs but also the learning process of the SNNs, they can also be considered to be meta-learning algorithms, which will be discussed in detail in the following chapter.

3

Meta-Learning

Although the effectiveness of machine learning has been proven in performing tasks in various fields, designing the appropriate training algorithm for specific tasks can be challenging. The learning algorithms are usually particularly designed for different tasks. Thus, transferring the algorithm to solve other tasks usually needs modifications and also needs to train the network again, which can be time-consuming. The demand for a robust machine learning algorithm led to the proposal of meta-learning. The idea of meta-learning was first introduced to the field of artificial intelligence by Schmidhuber (1987). Also known as learning-to-learn(l2l), meta-learning endows artificial neural networks the ability to learn, which allows the networks to learn new tasks rapidly without modifying the neural networks manually. Recent research has also extended the idea of meta-learning to the field of SNNs (Jordan et al., 2021). In this chapter, the general meta-learning methods will first be introduced in section 3.2. Then, meta-learning algorithms for SNNs will be discussed in section 3.3.

3.1 Basic Information of Meta-Learning

As mentioned above, meta-learning refers to the idea of learning to learn (Schmidhuber, 1987; Weng, 2018). By training the learner on a group of tasks, it can learn a new task outside the task group because of the learning skills obtained during the training. Meta-learning is sometimes confused with life-long learning. Life-long learning trains one model to perform a group of tasks, while meta-learning learns new models. Taking the example of a pattern classification task, the goal of machine learning is to learn a classifier f which maps the input pattern to its corresponding class. The goal of meta-learning is to learn a learning algorithm F, the output of which is the classifier f as described by Equation 3.1.

$$f = F(D_{train}) \tag{3.1}$$

For machine learning, the training dataset consists of inputs with or without labels. The training dataset for meta-learning is then the dataset for the group of tasks, and one dataset is one training sample. To avoid confusion, the training dataset for meta-learning is named the support set and the test set is named the query set (Weng, 2018).

Few-shot classification refers to meta-learning for supervised learning tasks. During the training process, n tasks are selected randomly from the support set and m samples are selected for each task. The learning is then called m-shot-n-class classification (Ravi and Larochelle, 2017). An example of a 4-shot-2-class image classification task is shown in Figure 3.1.

Recently, meta-learning has also been implemented in the field of reinforcement learning (Wang et al., 2016). The algorithms developed are usually known as Meta-RL. Meta-RL can be considered as two learning loops. At each iteration, the outer loop generates a new environment and adjusts the function or parameters which influence the agent's behaviour. The inner loop simulates the agent's behaviour in the environment and optimizes the accumulated reward. Different from the original reinforcement

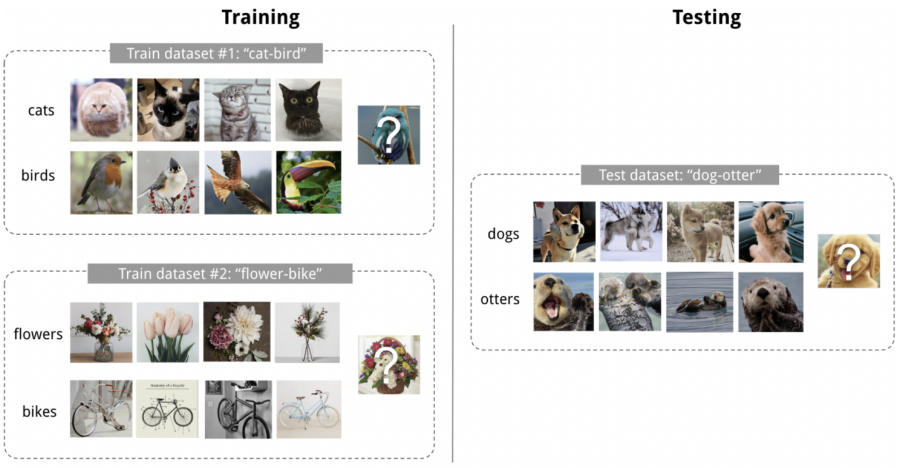


Figure 3.1: 4-shot-2-class classification (Weng, 2018)

learning, Meta-RL includes the reward and action at the previous timestep in the observation of the current step, acting as the history for updating the action strategy.

3.2 Common Approaches in Meta-Learning

Meta-learning is a broad field referring to an algorithm that learns the learning process of ANNs. Through the decades, researchers and scientists have developed various approaches to realize learning-to-learn. In the work of Huisman et al. (2021), they categorized meta-learning algorithms into three common methods: metric-based, model-based, and optimization-based. These three approaches will be introduced in this section respectively based on Huisman et al. (2021).

3.2.1 Metric-Based Meta-Learning

Metric-based meta-learning, also known as non-parametric meta-learning, is realized by learning the feature space for a group of tasks. The meta-learned feature will learn new tasks in a non-parametric way by comparing the new inputs to example inputs from the support set. The closer the prediction input is to the input example, the likelier that the prediction input has the same label as the example. Thus, the feature space for the metric-based meta-learning is also named the similarity kernel or attention mechanism, which computes the similarity between the test inputs and the example inputs. The probability distribution p_θ over classes Y of the output prediction of certain input is represented by the equation below:

$$p_\theta(Y|x, D^{tr}_{T_j}) = \sum_{(x_i,y_i)\in D^{tr}_{T_j}} k_\theta(x, x_i)y_i \tag{3.2}$$

where x is the input data from the query set and (x_i, y_i) are input data and label from the support set $D^{tr}_{T_j}$. The similarity kernel $k_\theta(x, x_i)$ computes the similarity between the prediction data and support data.

There are several implementations of metric-based meta-learning. For example, the Siamese neural network proposed by Koch (2015). consists of two twin networks with identical weights and network parameters. The Siamese neural networks learn a distance function to compute the difference between input data pairs. In the experiments carried out by Koch (2015), he first trained Siamese neural networks to learn discriminative features by comparing two input images in an image verification task. In the next stage, the model quickly learned to perform a new task with previously unseen input.

Metric-based meta-learning is favorable for non-parametric learning when the trained model is applied to new tasks, as no adjustments need to be made on the weights and parameters of the model. However, the defect of this method is its low performance when the distance between the test inputs and example inputs is large. Additionally, the computational cost for computing the similarity between pairs becomes huge when the dataset size increases.

3.2.2 Model-Based Meta-Learning

Model-based meta-learning, also known as black-boxes techniques, uses an internal architecture of the meta-learning controller to update parameters quickly. The neural networks process the input data in temporal order. The internal states of the model update with the input fed in at each time step. Inputs at each time step are remembered by a memorizer, which is essential for internal dynamics used to predict the output. The probability distribution for the prediction of a new input x can be described as:

$$p_\theta(Y|x, D_{T_j}^{tr}) = f_\theta(x, D_{T_j}^{tr}) \tag{3.3}$$

where f_θ is a neural network model with the connection weights θ.

A typical model-based meta-learning method is the recurrent meta-learners proposed by Duan et al. (2017), which applied a recurrent network functioning as the dynamic storage. Munkhdalai and Yu (2017) developed the meta networks that utilized another neural network learner to predict and generate weights of the model rapidly. Mishra et al. (2018) combines the temporal convolution and soft attention mechanism to serve as the memory storage.

As mentioned before, the core idea of model-based meta-learning is the memorizer for quick learning. Therefore, models with external memory storage are usually implemented. A widely used model is the memory-augmented neural network (MANN), which uses a storage buffer allowing the rapid encoding of new information and storing it for a long duration. An example of the implementation of the MANNs in model-based meta-learning was the work of Santoro et al. (2016). At each timestep of the training process, instead of feeding a pair of input data and its corresponding label, the input data at the current step and the label of the previous step are fed to the model. By doing so, the model is forced to memorize the input data and the stored information can later be retrieved for the prediction process. The backpropagation error signal during the prediction will be used to update the model weights.

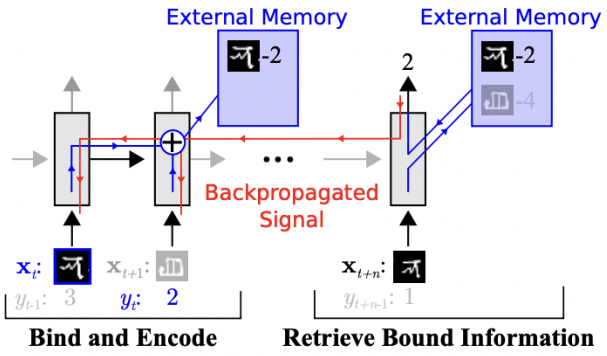


Figure 3.2: Learning scheme of model-based meta-learning with MANN adapted from Santoro et al. (2016)

Compared to metric-based meta-learning, model-based algorithms are more robust due to the flexibility of the internal dynamics. However, the performance of the model-based meta-learning in supervised learning is lower than that of the metric-based meta-learning. The performance also degrades when the data size increases. When the similarity between the new test tasks and the example tasks, model-based meta-learning also leads to unsatisfactory performances.

3.2.3 Optimization-Based Meta-Learning

Another common approach in meta-learning is optimization-based meta-learning. The optimization-based meta-learning usually consists of two learning loops, the inner loop learns to perform a specific task and the outer loop optimizes the performance in a task group. The goals of the outer loop are various for different algorithms. For example, some algorithms aim at optimizing the neural network architecture, while some may optimize the activation function. In general, any component of the algorithm that has the possibility to influence the performance can be considered as the target for optimization in this approach. Similarly to the other two approaches, the probability distribution p_θ over classes Y of the output prediction of certain input is represented as follow:

$$p_\theta(Y|x, D^{tr}_{T_j}) = f_{g_\varphi(\theta, D_{T_j}, L_{T_j})}(x) \tag{3.4}$$

where f is the inner loop learner, g_φ is the outer loop learner which optimizes f based on the parameters θ of f with support set D_{T_j} and loss function L_{T_j}.

There are various examples of applications of optimization-based meta-learning. LSTM optimizer proposed by Andrychowicz et al. (2016) learns the backpropagation update function by an RNN optimizer using a gradient-based method. As shown in Figure 3.3, the parameters θ of the optimizee neural network f referring to the inner loop learner are updated by the output g of the outer loop optimizer m. The optimizer m is a recurrent neural network, and its state h is updated by the gradient ∇ at each time step.

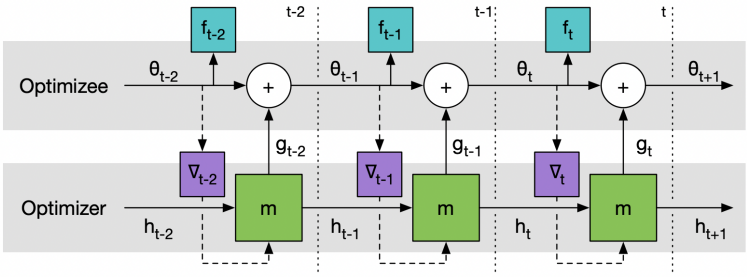


Figure 3.3: Learning scheme of LSTM adapted from Andrychowicz et al. (2016)

LSTM meta-learner is another algorithm that learns the weight update function, but it embeds the weights into the cell state and updates the weights by updating cell states (Ravi and Larochelle, 2017). Yuxi (2017) applied a reinforcement approach to learn the optimizer which is in the form of a recurrent neural network. Another popular optimization-based meta-learning method is the model-agnostic meta-learning (MAML) algorithm proposed by Finn et al. (2018). The outer loop in MAML learns to set a good initialization for each task learned in the inner loop.

The advantage of optimization-based meta-learning is that it is more robust compared to metric-based and model-based methods. For a task group with larger diversity, the optimization-based approach showed better performance (Finn and Levine, 2018). However, the disadvantage is the high computational cost for learning the algorithm for each task.

3.3 Meta-Learning in Spiking Neural Networks

Recently, the concept of meta-learning has been applied in the field of SNNs. Aside from enhancing the robustness of the models by allowing rapid adaptation, meta-learning can also be used to optimize the learning algorithm. Due to the difficulty in training SNNs, meta-learning has been considered as an approach to increase the learning capacity of SNNs in recent years. Most research in meta-learning of SNNs belongs to the category of optimization-based meta-learning, where the weight update rules are learned for training the SNNs. In this work, the algorithms are divided into two types based on

the format of the learning rule optimizer: symbolic approach and ANN approach. The former utilizes symbolic expressions to describe the weight update function and the latter applies neural networks to compute weight update. Examples of both approaches will be introduced in this section.

3.3.1 Symbolic Approach

The earliest examples of evolving the weights update were mainly based on a parameterized Hebbian rule in the following format (Niv et al., 2002; Toenlli and Mouret, 2011):

$$\Delta W = \eta(Ax_ix_j + Bx_i + Cx_j + D) \tag{3.5}$$

where x_i and x_j are the pre- and post-synaptic neural activities. Coefficients $A - D$ and learning rate η are parameters to be evolved. Although boosted performances have been observed, the search space was very limited due to the specified Hebbian rule format. Thus, additional solutions with other formats were excluded using these methods. However, in the research of Jordan et al. (2021), the update rules to be evolved are not limited to a preliminarily defined formula. The coefficients, operands, and operators are all genomes to be evolved. The plasticity can be expressed by a general format: $\Delta W = \eta f(...)$, where $f(...)$ is the function to be evolved. Additionally, the generality of this method was proven by experiments in three different types of tasks: reinforcement learning tasks (reward-driven tasks), supervised learning tasks (error-driven tasks), and tasks where correlations between pre- and postsynaptic activity are available.

The strategy can be summarized into three steps:

1. Determine a task family of interest and associated network architecture (neuron types and connectivity).
2. Define the fitness function from examples from the given task family.
3. Specify the input signals available to the plasticity rule, such as traces of pre- and postsynapsess.

The fitness function is simply expressed by the average cumulative reward obtained over n experiments. The reward R is 1 if the classification is correct and -1 otherwise. They chose the input variables for the weight update rule based on prior knowledge in the learning of this type of task. A previous research by Urbanczik and Senn (2009) discovered a learning rule as expressed in Equation 3.6. The plasticity rule only depends on the reward R and synapse trace E and has been proven to learn pattern classification tasks effectively.

$$\Delta W = \eta(R - 1)E \tag{3.6}$$

Thus, the same variables (E, R) were selected as the inputs to the evolutionary algorithm. In about half of the total generations, the evolutionary algorithm evolved a plasticity rule identical to the one derived in previous work in most cases. Some runs also evolved variations of the learning rule with good learning performance. Similarly, the experiments in the other two types of tasks also used the prior knowledge in the learning rules of the specific tasks to determine the input variables. For both cases, they successfully evolved efficient learning rules identical to the prior derived learning rules and other variations which lead to comparable performance.

Symbolic Expression Evolution Method

The meta-learning of learning rules with the symbolic approach usually involves the evolution of a symbolic expression. There are various algorithms for evolving symbolic expressions. Three commonly used methods will be introduced below: gene expression programming (GEP), grammatical evolution (GE), and cartesian genetic programming (CGP).

Gene Expression Programming (GEP)

Gene Expression Programming (GEP) uses a linear string of fixed length to express the chromosomes, which is named K-expression (Oltean et al., 2009). The K-expression is then mapped into expression

trees as shown in Figure 3.4. The conversion starts from the first position in the string, which corresponds to the root of the expression tree, and reads through the string one by one. For each node in one layer in the tree, if it is a function with $n(n \geq 1)$ arguments, then the next n symbols in the string are attached below it as n child branches. Otherwise, the terminal node will become a leaf of the expression tree.

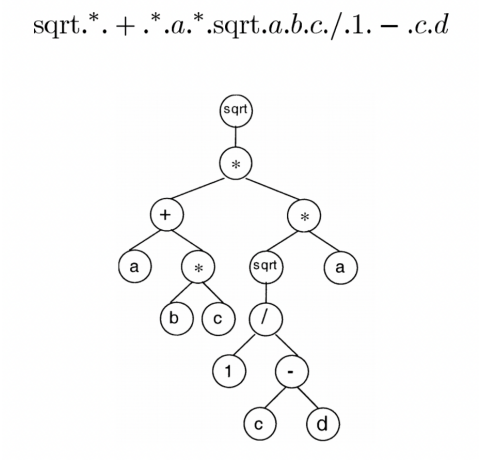


Figure 3.4: K-expression and the corresponding expression tree adapted from Oltean et al. (2009)

The string is divided into a head and a tail (Oltean et al., 2009). The head contains both operators and operands, while the tail only contains operands. The length of head h is a parameter determined by the user and the tail length t is a function of h and the maximum arity of functions n: $t = (n-1) * h + 1$. In the case of multigenic chromosomes, the genes will be described by sub-expression trees and linked by a "+" function to form the final expression tree. The multigenic chromosomes can also be composed by including a homeotic gene, which describes how the sub-expression trees interact with each other. New individuals can be generated by crossover, mutation, and transposition.

Grammatical Evolution (GE)

Grammatical evolution (GE) uses a domain-specific language (DSL) such as Backus-Naur Form (BNF) to describe mathematical expressions (Oltean et al., 2009). There are many different grammars developed for GE, a relatively simple grammar designed by Bello et al. (2017) to learn optimizers for reinforcement learning will be introduced in detail. In Bello's work, a string is constructed with certain components. The string consists 1) the first operand to select op_1, 2) the second operand to select op_2, 3) the unary function to apply on the first operand u_1, 4) the unary function to apply on the second operand u_2, and 5) the binary function to apply to combine the outputs of the unary functions b. The output of the binary function is then either temporarily stored in the operand bank (can be selected as an operand) or used as the final learning rule. The mathematical expression and visualization of the algorithm are shown in Equation 3.7 and Figure 3.5 respectively.

$$\Delta W = \eta \times b(u_1(op_1), u_2(op_2)) \tag{3.7}$$

Their experiments demonstrated that with a limited number of iterations, the DSL can only represent a subset of all mathematical equations. However, it can represent common optimizers within one iteration assuming access to simple primitives such as gradients or bias-corrected running estimates of the gradients.
Bello et al. (2017) used a Proximal Policy Optimization (PPO) algorithm to learn the optimizer while Alber et al. (2018) used an evolutionary approach. The evolutionary controller maintains a fixed population N. Initially, the controller samples N equations at random from the search space. At each iteration, the evolutionary controller does one of the following:

1. With probability p, the controller chooses an equation randomly at uniform within the N best candidates found so far during the search.

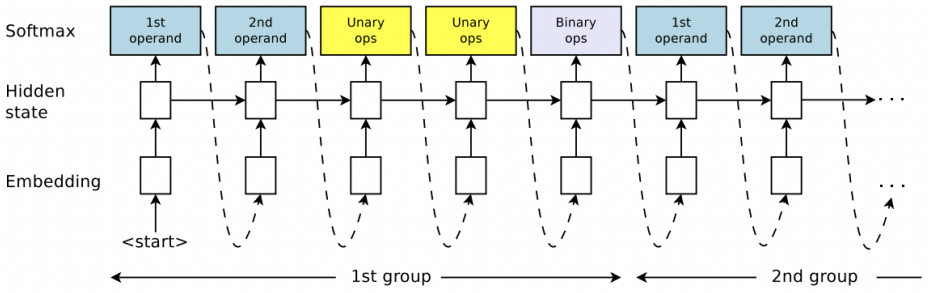


Figure 3.5: Overview of the grammatical evolutionary controller adapted from Bello et al. (2017)

2. With probability $1-p$, it chooses an equation randomly at uniform from the rest of the population.

The controller applies k mutations to the selected equation. Each of these k mutations simply selects one of the components (an operand, a unary function, or a binary function) at random and replaces it with another component of the same category chosen uniformly at random. Certain mutations lead to mathematically infeasible equations, when this is the case, the controller restarts the mutation process until a feasible equation is found. An advantage of this method is that it can start from existing equations (Alber et al., 2018).

Cartesian genetic programming (CGP)

Cartesian genetic programming (CGP) uses an indexed graph representation of programs (Oltean et al., 2009). The genotype of an individual is a 2D Cartesian graph with a fixed number of input, output, and internal nodes. The nodes of the inputs indicate the index of the input. Each internal node has a string of three integers: the first integer indicates the operator and the other two integers indicate the index of nodes to be connected. The output of the internal nodes can also be used as input to other internal nodes.

The genes of a CGP are randomly initialized but several constraints should be designed to ensure the validity of the chromosome. Mutation can occur in the operator and the connection of the nodes. However, the mutation operator should also follow the constraints for the initialization. An example of the CGP is shown in Figure 3.6, the internal nodes have two columns and three rows. The index and the corresponding operators or operands are listed in the function table. The mutation example shows both the mutation in nodes connection and the mutation in function gene.

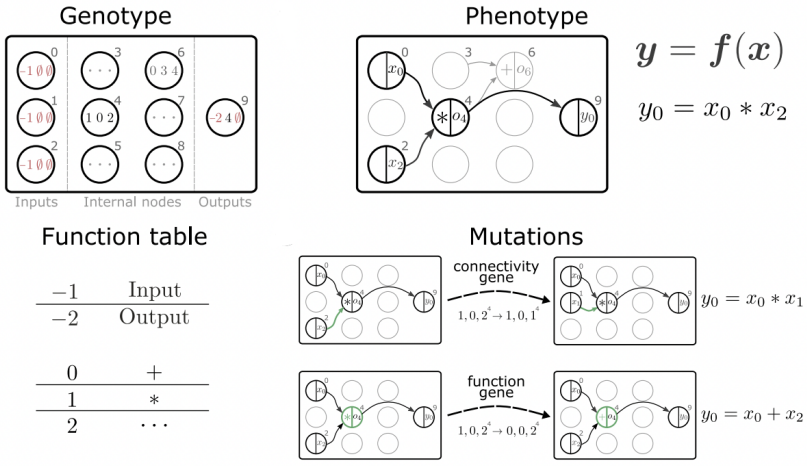


Figure 3.6: Visualization of Cartesian Genetic Programming adapted from [Jordan et al., 2021]

3.3.2 ANN Approach

Aside from learning symbolic expressions of plasticity rules, more meta-learning solutions in SNNs evolve ANNs for weight update. An algorithm with high influence is the Adaptive HyperNEAT developed by Risi and Stanley (2010). The origin version is the HyperNEAT, which uses a compositional pattern producing network (CPPN) to generate a connection pattern along with weights within the geometry of the ANNs.

Adaptive HyperNEAT extends HyperNEAT by including the patterns of plasticity rules. By adjusting the architecture of CPPNs, the algorithm can generate both a general but noninterpretable weight update and a less general but interpretable plasticity rule. The general Adaptive HyperNEAT model is named the iterated model. Aside from the location of each pair of nodes in the neural networks (x, y), presynaptic activity o_i, postsynaptic activity o_j, and the current connection weight w_{ij} are all used to update weights as shown in Equation 3.8.

$$\Delta w_{ij} = CPPN(x_1, y_1, x_2, y_2, o_i, o_j, w_{ij}) \tag{3.8}$$

The less general HyperNEAT model is called the Hebbian ABC model, which is based on a parametric Hebbian formula. The inputs to the CPPN remain the same as the original HyperNEAT but the output of CPPN should also generate the learning rate and coefficients of the Hebbian learning rule as expressed by Equation 3.9. The demonstration for both models are visualized in Figure 3.7.

$$\Delta w_{ij} = \eta(Ao_1oj + Bo_i + Co_j) \tag{3.9}$$

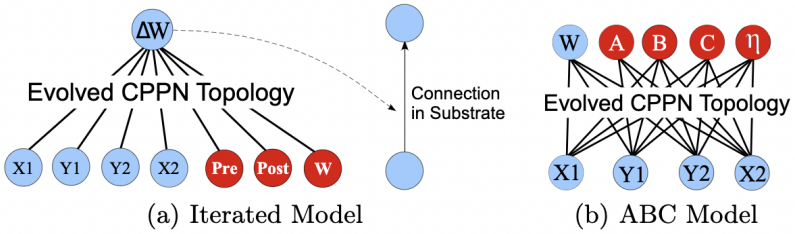


Figure 3.7: Visualization of two HyperNeat models adapted from [Risi and Stanley, 2010]

The experiments performed in Risi and Stanley (2010) trained networks to perform the T-maze problem with two scenarios. The first scenario had a linear reward signal, and the second scenario had a nonlinear reward signal. The ANN meta-learner is a simple two-layer neural network with three input neurons and three output neurons. It was found that in the scenario with linear reward, both the iterated and ABC models were able to reach the target fitness after a similar number of generations, but the ABC model evolves faster in the beginning. However, in the scenario with nonlinear reward, the ABC model failed to meet the target model since it could not evolve a nonlinear learning rule. Although the iterated model evolved slowly, it is able to perform the task well in the end. Thus, the authors concluded that while the ABC model with a properly designed ANN topology should be sufficient for most domains, the iterated model is more robust to different scenarios even with a simple ANN topology. However, Risi and Stanley (2010) also mentioned that the iterated model is computationally expensive because the CPPN must be requeried for every ANN connection every time step. It might be too computationally expensive for tasks requiring large neural networks.

Instead of evolving both the connection pattern and the weight update, more examples have a designed neural network architecture and another ANN to be evolved for computing the weight update. An example is the research of Orchard and Wang (2016), where they used a three-layer recurrent and fully connected neural network (4 input neurons, 3 or 5 hidden layer neurons, 2 output neurons) to perform a foraging task. The genomes to be evolved are the initial weights and bias of the network and connection weights. For comparison, they also did the same experiments using the approach of evolving the parametric Hebbian update rule. The experimental results demonstrated that the approach based

on the parametric Hebbian rule learned faster and better in a certain number of timesteps. They also extended the lifetime of each run and applied a sensory offset that varies every 1000 steps. In this case, the learning of the Hebbian update rule was similar to that of the ANN update but stalled at a certain number of action steps. Thus, they concluded that the ANN update rule allows the neural controller to adapt to changing circumstances.

Another example is the implementation of meta-learning in neuromorphic hardware (Bohnstingl et al., 2019), where they applied the idea of learning-to-learn (L2L) with an inner loop and an outer loop. The outer loop evolves the hyperparameters of the neural network model in a family of tasks and the inner loop learns one task from the family using the evolved model. The task family they chose was a simple reinforcement learning task: multi-arm bandit problem. Similar to the previous example, they also did the experiments using two approaches: one is evolving the coefficients of the parametric Q-learning rule and the other is evolving an ANN to update weights. Performing both experiments in the neuromorphic hardware, they demonstrated that learning using an ANN for the weight update achieved better performance.

4

Synthesis of Literature

After an overview of the related topics has been given, a summary of the most relevant content will be presented in this chapter. For the preliminary experiments, the high-level design decisions will also be discussed based on an analysis and comparison of the candidate solutions. Section 4.1 will draw attention to the neuron model. The properties of different neuron models will be summarized and compared. Then, section 4.2 will provide synthesized information on the learning approaches presented in section 2.3. Section 4.3 focuses on the state-of-art meta-learning algorithms for the SNNs followed by a trade-off between the symbolic approach and ANN approach. In the end, the expression methods for learning rules will be compared in section 4.4.

4.1 Neuron Model

The neuron model is the unit component of SNNs, which mimics the behaviours of biological neurons in the process of signal transmission. As discussed in section 2.2, there are various neuron models available for the SNNs. The neuron models are derived based on experimental data of communication in natural nervous systems. The signal transmission between biological neurons is a complex process involving a number of neuronal behaviours (Gerstner et al., 2014). The derived neuron models describe a part or all of the behaviours. For examples mentioned in section 2.2, the HH model exhibits almost all the behaviours that have been discovered in signal transmission between neurons while the I&F model and Izhikevich model only describe a few neuronal behaviours (Abusnaina and Abdullah, 2014). The models that exhibit more behaviours mimic the biological nervous systems more closely, and therefore, have higher fidelity. However, it usually comes with the cost of larger computational consumption, which can be unfavorable for training SNNs on complex tasks. Thus, it is important to choose the appropriate model for the research based on the objective and settings of the experiments.

A comparison of the complexity and fidelity of the models are shown in Table 4.1. The complexity is represented by the number of floating-point operations and variables and the fidelity is represented by the number of behaviours that the models can describe.

Based on the comparison, the LIF model is selected for this work. Since the objective of this research is to evolve learning rules for training SNNs automatically, the process of learning-to-learn will consume large computational power. Additionally, the research also aims to increase the evolutionary search space and the complexity of the SNNs architecture and tasks, which will further increase the computational cost. Therefore, a neuron model with low complexity is favorable for this work. Furthermore, high-fidelity regarding the biological neural system is not a critical factor in this work. Thus, the LIF model is sufficient to describe the neuron model in this research. Furthermore, the neuron model in the Loihi chip is also the LIF model (Davies et al., 2018). Applying the same neuron model is helpful for the later implementation of the algorithm on hardware.

Model	Number of floating point operations	Number of variables	Number of behaviours
LIF	6	1	3
Izhikevich	13	2	20
BVP	72	2	11
Wilson	180	2	14
Morris-Lecar	600	3	14
HH	1200	1	21

Table 4.1: Comparison between neuron models. [Abusnaina and Abdullah, 2014]

4.2 Learning Approach

Similar to ANNs, SNNs can be trained with unsupervised learning, supervised learning, reinforcement learning, and evolutionary algorithms. In most cases, the learning algorithms for SNNs are based on the synaptic plasticity rules, which describe the variation of synaptic connection strength with spike activities of pre- and postsynaptic neurons. Hebb's rule first proposed the theory of "cells that fire together wire together" (Hebb, 1949). After the discovery of LTD and LTP phenomena, the STDP rule was developed, which exhibits the influence of the temporal order of the pre- and postsynaptic spikes (Markram et al., 1997). Synaptic plasticity rules allow the SNNs to learn unsupervised tasks by correlations between spikes of neuron pairs (Diehl and Cook, 2015; Guo et al., 2019). They can also be used for learning supervised tasks by introducing teaching signals (Ruf and Schmitt, 1997). A closer imitation of the learning behaviour in nature is realized by modulating the synaptic plasticity rules by rewards received from the environment. The reward modulated synaptic plasticity rules also allow the reinforcement learning of SNNs (Florian, 2007). Neuroevolution is a broader field, which applies evolutionary algorithms in training SNNs. It can be used to evolve connection weights without plasticity learning rules (Hagenaars et al., 2020; Qiu et al., 2018).

The selection of learning approaches is usually performed based on the target type of tasks. Since this research aims at evolving learning rules, learning without plasticity rules such as evolving connection weights directly will be dismissed. Furthermore, the target type of task is continuous control tasks, which are usually solved by reinforcement learning. As a result, the reinforcement learning approach will be implemented to train SNNs in this work. In other words, this research will evolve reward-modulated plasticity rules for training SNNs.

4.3 Meta-Learning Approach

There have been several meta-learning algorithms developed in the field of SNNs, which have the goal of optimizing learning rules for training SNNs. Based on the expression methods for the learning rules, these algorithms are categorized into two approaches: symbolic approach and ANN approach. The former describes the learning rules with mathematical expressions. Most examples evolve the coefficients of the Hebbian learning rule (Niv et al., 2002; Toenlli and Mouret, 2011). There is also research that evolves the entire learning rule by symbolic expression (Jordan et al., 2021). The other approach expresses the weight update function as an ANN instead of a mathematical formula (Risi and Stanley, 2010; Orchard and Wang, 2016; Bohnstingl et al., 2019). The learning of SNNs is optimized by evolving the ANN learner.

The advantages and disadvantages of both approaches will be discussed. The most important feature of the symbolic approach is that the evolved learning rules are interpretable expressions, which is favorable for understanding how the network is shaped during the learning and human-guided generalization in different task domains (Jordan et al., 2021). Besides, the symbolic approach which evolves coefficients

of a fixed learning rule is usually simple and less computationally expensive (Bohnstingl et al., 2019). Also, experiments have proven that the algorithms have achieved good learning performances in various tasks (Niv et al., 2002). However, a disadvantage of this approach is that learning symbolic expressions usually requires prior knowledge (Jordan et al., 2021). Methods that evolve parameters have limited search space compared to the ANN approach. Additionally, the learning performance can be influenced by the design of the neural networks (Bohnstingl et al., 2019).

Compared to the symbolic approach, the ANN approach is more robust to the neural network architecture and more generalizable to different task domains (Risi and Stanley, 2010). Experiments that applied both approaches demonstrated that the ANN approach could achieve comparable or even better performances than the symbolic approach. Additionally, the ANN approach is less limited by prior knowledge. The main drawback of this approach is that it is more computationally expensive even for simple tasks with small neural networks. Furthermore, since the weight update is computed by the ANN directly, the learning procedure is difficult to analyze (Jordan et al., 2021).

The algorithm proposed by Jordan et al. (2021) can be considered as a middle ground between both approaches. Since this algorithm evolves the entire expression of the learning rule, it has a larger search space than methods that only evolve coefficients. Therefore, it is also more robust and flexible. Besides, the result of evolution is a mathematical expression of the learning rule. Thus, the algorithm has the potential to reveal the learning scheme of SNNs on a certain type of task. As a result, this specific meta-learning method will be implemented in this research.

4.4 Symbolic Expression

Since the selected meta-learning algorithm evolves the mathematical expression of the learning rule, it is also necessary to choose an appropriate symbolic expression method. Section 3.3.1 introduced three typical symbolic expression methods: gene expression programming (GEP), grammatical evolution (GE), and cartesian genetic programming (CGP). GEP describes a formula by a linear string that can be mapped into an expression tree. GE refers to the expression methods that construct functions by a domain-specific language, where the functions can be represented by a string or a list of numbers. CGP represents the mathematical expression by using an indexed graph. The integers in the graph represent the operators or operands.

Each expression method has its own advantages and disadvantages, and the selection of expression method can be based on two criteria according to Oltean et al. (2009): syntactic validity and expression capability. By dividing the chromosome into a head and tail, GEP can generate syntactically correct math expressions efficiently. CGP requires a set of properly designed constraints for initialization and mutation to generate valid output. GE allows chromosomes encoding invalid expressions, thus an extra program for validity check is needed. Regarding the expression capability, it can also be evaluated by the maximum length of the functions. The number of terms of the function is limited by the designed length of string for GEP. Similarly, the maximum number of terms is also fixed by the number of columns of the phenotype graph in the CGP approach. However, the search space for GE is much larger since it has fewer limitations in mapping and the output of one generation can be used as the input for next generations. Thus, prior knowledge of the format of the target formula can help the user determine the design parameters when using CGP and GEP and narrow down the search space. However, the program may fail to generate the target formula if the design parameters are not properly selected. GE is suitable for evolving expressions with little prior knowledge but it may require a large number of generations to learn the target function. An additional advantage of GE is that it allows learning from bootstrap by including half-evolved expressions in the initial operand bank.

CGP will be implemented in this work since it is relatively simple and requires less computational power. Moreover, the output of CGP is syntactically valid and it eliminates the very-long learning rules by setting a limit on the number of terms. Thus, it automatically eliminates candidates that are invalid or intricate.

III

Preliminary Experiments

5

Methodology

The objective of the preliminary experiments is to understand the scheme of evolving biological plausible learning rules for SNNs. By investigating different components of the experiments such as the SNN architecture and input to the evolutionary search, a better knowledge of the necessary prerequisite for answering the research question can be obtained. The preliminary experiments started from a rough reproduction of the reinforcement learning task in Jordan et al. (2021) and further extended to other simple tasks and varied experiment settings. By the end of the preliminary experiments, a preliminary understanding of the research topic should be obtained.

In this chapter, the outline of the preliminary experiments will be presented in section 5.1. The task description and experiment settings for each experiment will be discussed in section 5.2, section 5.3, and section 5.4.

5.1 Outline of the Analysis

Since this research is inspired by the evolving-to-learn algorithm developed by Jordan et al. (2021) and can be considered an extension of their work, it is natural to start by reproducing their experiments to obtain a comprehensive understanding of the principle of their algorithm. In their work, three experiments were performed: evolving reward-modulated synaptic plasticity rules in a reinforcement learning task, evolving error-driven synaptic plasticity rules in a supervised learning task, and evolving correlation-driven synaptic plasticity rules in an unsupervised learning task. Although the experiment details such as the SNN architecture vary for each task, the main settings and principles such as the evolutionary strategy and symbolic expression method are the same. Thus, it should be sufficient to select one of the tasks for reproduction. Since the final goal of this research is to evolve learning rules for training SNNs to perform continuous control tasks, which are usually learned with reinforcement learning methods, the reinforcement learning task among Jordan's three experiments is more relevant to this work. Therefore, the first preliminary experiment is reproducing the reinforcement learning task in Jordan et al. (2021), which is a binary random pattern recognition task.

In the second experiment, the task is redesigned so that the input patterns have a meaningful difference between the classes. The original task uses random Poisson patterns, which is less meaningful. Additionally, the randomness of the input pattern also increases the difficulty in analyzing the evolved learning rules. Thus, the patterns used in the second experiment were classified based on the temporal order of their spikes while the general settings remained the same as the first experiment.

The third experiment is a further extension of the original experiment. In Jordan's work, the evolution of learning rules was based on the prior knowledge of specific tasks. In other words, they had a baseline learning rule that had been proven to be able to train the SNNs on the task successfully and the inputs to the evolutionary search were also terms taken from the baseline learning rule. In the third preliminary experiment, attempts are made to evolve learning rules without prior knowledge.

5.2 Evolving Learning Rules for a Random Pattern Classification Task

5.2.1 Task Description

Since the aim of the first preliminary experiment is to reproduce the work of Jordan, the task was set to be the same as in Jordan et al. (2021). For each experiment trial, 30 random Poisson spike patterns are generated. Each pattern consists of N spike trains with a duration of T timesteps. Thus, each pattern has a shape of (N, T), where N was set to be 50 and T was set to be 500 ms. The spiking rate is 6 Hz for all spike trains resulting in an average spike number of 3 for each spike train. The 30 patterns were assigned a label of 0 or 1 randomly. For each experiment trial, the spiking neural network is trained to learn classifying the patterns by updating the weights for 200 epochs. The classification is done by counting the total number of output spikes, the prediction is 1 if the total number of output spikes is higher than the average and the prediction is 0 otherwise.

5.2.2 Neuron Model and Spiking Neural Network Structure

The neuron model used in Jordan et al. (2021) for this task was a LIF model with exponential post-synaptic currents. Compared to the standard LIF model, it describes the current input to the post-synaptic neuron more accurately and is thus closer to the signal transmission in biological neurons. However, introducing an exponential term also increases the complexity of the model and requires more computational power. Since the evolutionary search already requires a large amount of computational power, a simpler neuron model is preferred. As a result, a standard LIF model will be used in the preliminary experiments instead and the refractory phase of the neuron is ignored.

As discussed in section 2.2, the LIF model can be represented by a differential equation as shown in Equation 5.1. The differential equation can be further interpreted as equations for updating the neuron states at each time step. The neuron states to be updated are current I, membrane potential V, and the indicator of spike Z. They are computed by Equation 5.2, Equation 5.3, and Equation 5.4 respectively. The current is accumulated by the input, which is the multiplication of the synapse weights and input spikes. The membrane potential is updated by the current. The term $(1 - Z)$ describes the phase when the potential drops to zero after the neuron releases a spike. The constants τ_I and τ_V describe the decay of current and membrane potential with time. Aside from the states in the LIF model, another neuron state usually involved in learning is the eligibility trace E of the neuron. It is usually updated at each time step by Equation 5.5. The equation describes the trace of the spikes f, which can be either the presynaptic spike or postsynaptic spike. The constants τ_e and α are the trace decay and amplitude.

$$C_m \frac{dV}{dt} = -\frac{V - E_m}{R_m} + I \tag{5.1}$$

$$I_{new} = I\tau_I + (1 - \tau_I)I_{input} \tag{5.2}$$

$$V_{new} = V\tau_V(1 - Z) + (1 - \tau_V)I_{new} \tag{5.3}$$

$$Z_{new} = \begin{cases} 1 & if \quad V_{new}\theta \\ 0 & else \end{cases} \tag{5.4}$$

$$E_{new} = E * \tau_e + \alpha f \tag{5.5}$$

Since deriving the topology of the SNN is not the goal for the evolution, the architecture of the network is designed manually. A simple SNN with two layers is implemented. The first layer is an input layer consisting of 50 neurons. The input neurons are Poisson generators without neuron states and characteristics. The output layer has only one output neuron. The output neuron is a LIF neuron

described by the model given above. Compared to other SNNs with multiple layers of LIF neurons, the simple architecture applied in the experiment reduces the computational cost but increases the difficulty of learning in the meantime. The task is visualized in Figure 5.1.

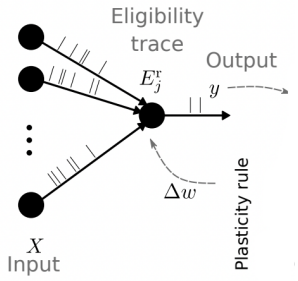


Figure 5.1: Visualization of the random pattern classification task adapted from Jordan et al. (2021)

5.2.3 Genetic Programming for Evolutionary Search

The genetic programming method implemented in this work to search for effective plasticity learning rules is the Cartesian Genetic Programming (CGP) method, which is the same as the one used in Jordan et al. (2021). The plasticity rule for training the SNNs has the form of $\Delta W = \eta f(...)$. The learning rate η is pre-tuned and fixed during the evolution. It is also possible to include the learning rate into the evolutionary search, however, the computational cost will also increase. The function $f(...)$ is the mathematical expression to be evolved. The variables to the function are specified from the baseline plasticity rule. The baseline function is the plasticity rule derived by Urbanczik and Senn (2009), where the weight update only depends on the synapse trace and reward.

As already discussed in subsection 3.3.1, the genotype of the individuals is represented by a 2D indexed graph. The phenotype is then a list of indexes that can be interpreted into a mathematical expression. The elements of the indexed graph are the two variables, four basic operators $(+, -, \times, \div)$, and a constant float. The number of columns and rows of the indexed graph is set to be the same as that in Jordan's work, which is 12 and 2 respectively. Thus, the evolved plasticity rule can be relatively long, allowing more space for the evolutionary search.

Since the weight initialization is performed randomly for each experiment trial, there can be cases that the initialization is coincidentally near an optimal solution resulting in a good performance, and also cases where bad initialization brings difficulties for training. Thus, each individual performs three trials of the experiment in the evolutionary search. The fitness function is defined to be the average of the test accuracy of the three trials. The individuals with higher fitness are selected based on the $\mu + \lambda$ algorithm. μ is the number of parents and λ is the number of offspring mutated from each parent. In each generation, the two individuals with the highest fitness are set to be the parent for the next generation. The breed is realized by random mutation in the phenotype. In other words, the index of the index list will be replaced randomly. Thus, both the operators and operands can be mutated.

5.2.4 Pseudo Code

The algorithm of the task is described by the pseudo-code below.

```
Initialise parents candidate solutions and their offspring
loop for each individual
    while number of trials not met do
        Initialise weights
        loop for each epoch
            loop for each pattern
                Reset neuron states and synapse trace
                loop for each time step
                    Feed input to the SNN
                    Update neuron states and synapse trace
                end loop
                Compute reward
                Update weights with the candidate solution
            end loop
        end loop
    end while
    Compute fitness
end loop
Select new parents and breed offspring for the next generation
```

5.3 Evolving Learning Rules for a Temporal Order Classification Task

5.3.1 Task Description

In the second experiment, the content of the task is different but most experiment parameters are kept the same: each trial has 30 patterns with 2 classes and each pattern has a shape of (50, 500). However, instead of generating random spike patterns and labels, the patterns are classified by the temporal order in this task. For the first class with the label of 0, the input neurons spike in order, the first neuron spikes first and the last neuron spikes last. For the second class with the label of 1, the input neurons spike in the opposite order, the last neuron spikes first and the first neuron spikes last. Each input neuron only spikes once and the exact time of spike is not fixed. Thus, the input spike trains can be different for the 30 patterns.

5.3.2 Experiment Setup

The experiment setup is similar to that of the first experiment. The SNN also has 2 layers: the input layer with 50 spike generators and the output layer with only one LIF neuron. For the evolutionary search, the same CGP parameters are used. The input variables to the CGP are also the synapse trace and reward.

5.4 Evolving Learning Rules for an XOR Task without Prior Knowledge

5.4.1 Task Description

The evolutionary search in the experiments before was based on prior knowledge, the SNNs were first trained with a baseline learning rule and the hyperparameters were tuned before the evolutionary search. To investigate whether the algorithm can evolve effective learning rules without pre-training, the algorithm is implemented in a different pattern classification task. The task is selected to be the XOR pattern classification, which has a comparable difficulty level with the previous tasks. The XOR problem has four patterns with two labels: (0,0) with the label of 0, (0,1) with the label of 1, (1,0) with the label of 1, and (1,1) with the label of 0. The integer inputs are converted to spike trains by

rate encoding. 0 corresponds to spike trains with a spiking rate of 10 Hz, while 1 corresponds to spike trains with a spiking rate of 50 Hz. Each spike train has a total duration of 100 time steps. Thus, each input pattern has a shape of (2, 100). For each experiment trial, 30 inputs are generated by randomly selecting spike trains from the four patterns.

5.4.2 Experiment Setup

The SNN designed for this task has three layers: an input layer with 2 neurons, a hidden layer of 5 neurons, and an output layer of 1 neuron. Similar to other experiments, the input neurons are spike generators without neuron characteristics while other neurons have the LIF characteristics. Different from previous experiments where the input variables to the evolutionary search are taken from the baseline learning rule, this experiment is independent of prior knowledge of learning rules that can solve the task. Since the goal of this project is to expand the search space of the evolutionary search, more input signals are made available:

- Presynaptic traces with slow and fast decays: x_1, x_2
- Postsynaptic traces with different decay constants: y_1, y_2, y_3
- Binary variable indicating the spike of the pre- and postsynaptic neuron: x_0, y_0
- Reward trace: r_1

For each synapse, multiple presynaptic traces and postsynaptic traces with different decay parameters are available.

6

Experimental Results

The results of the three preliminary experiments will be discussed in this chapter. For the experiments pre-trained with prior knowledge, both the results of the SNNs trained with the baseline learning rule and the evolutionary search will be discussed in section 6.1 and section 6.2. In section 6.3, the results of the evolutionary search for the last experiment will be discussed.

6.1 Random Pattern Classification

6.1.1 Baseline Learning Rule

The reproducing of the experiments of Jordan started from training the SNNs on the random pattern classification task with the baseline plasticity rule. The reward-modulated plasticity rule utilized by Jordan is the Urbanczik rule derived in the work of Urbanczik and Senn (2009). Urbanczik and Senn developed a reinforcement learning scheme that only depends on the synapse trace and reward achieved at the end of each episode, which is expressed in Equation 6.1. R is the reward obtained each episode and R_{base} is the reinforcement baseline that balances reinforcement and punishment. In Jordan's work, R_{base} was set to 1, R was 1 for a correct prediction, and -1 for a wrong prediction. Using the same plasticity rule as designed by Jordan, it was found that the connection weights decreased with training epochs in general due to the reinforcement baseline. Thus, it requires large initial weights, which is different from common learning approaches. Since the Urbanczik rule performs weight updates at the end of each episode, in this case, the plasticity rule will be applied at the end of the time step for each training cycle. As a result, integration of the synapse trace is required to describe the trace over the episode duration. Performing the integration at each time step also requires more computational cost. Additionally, the prediction of patterns in Jordan's work is defined as follows: the prediction is 0 if the total number of output spikes is 0, the prediction is 1 if the total number of output spikes is larger than 0. A drawback of this prediction method is that the 0 output spike for patterns with label 0 may lead to small weights. In case the weights are too small, the neural networks may be inactive and fail to learn due to zero trace.

$$\Delta W = \eta(R - R_{base})E(T) \tag{6.1}$$

Because of the above issues revealed in the early attempts of the reproduction task, other baseline learning rules were investigated. Since the RSTDP rule developed by Florian (2007) has proven its ability in learning simple pattern classification tasks, it was selected and implemented in this experiment. As mentioned in section 2.3, Florian derived two plasticity rules: MSTDP for learning without eligibility trace, MSTDPET for learning with eligibility trace. MSTDPET rule was implemented in this work since it was found to have a slightly better performance in learning pattern classification tasks. The plasticity

rule consists of five equations as mentioned in section 2.3:

$$w_{ij}(t+\delta t) = w_{ij}(t) + \gamma \delta t r(t+\delta t) z_{ij}(t+\delta t) \tag{6.2}$$

$$z_{ij}(t+\delta t) = z_{ij}(t) + \xi_{ij}(t)\exp(-\delta t/\tau_z) \tag{6.3}$$

$$\xi_{ij}(t) = P_{ij}^{+}(t) f_i(t) + P_{ij}^{-}(t) f_j(t) \tag{6.4}$$

$$P_{ij}^{+}(t) = P_{ij}^{+}(t-\delta t)\exp(-\delta t/\tau_{+}) + A_{+} f_j(t) \tag{6.5}$$

$$P_{ij}^{-}(t) = P_{ij}^{-}(t-\delta t)\exp(-\delta t/\tau_{-}) + A_{-} f_i(t) \tag{6.6}$$

Different from the Urbanczik rule, the MSTDPET rule updates weights every timestep. Thus, the reward should also be assigned to the network at each timestep. Therefore, the prediction can not be performed at the end of the simulation duration and a new prediction method should be defined. The new prediction method was inspired by the experiment in classifying temporal coded XOR patterns performed by Florian. If the label is 0, the reward is -1 for each spike released. If the label is 1, the reward is 1 for each spike released. In case of no output spike, the reward is 0 for both labels. An example of a learning process for a pattern with label 1 is shown below. As shown in Figure 6.1, since the label is 1, each release of the output spike f_i will result in a reward of 1.

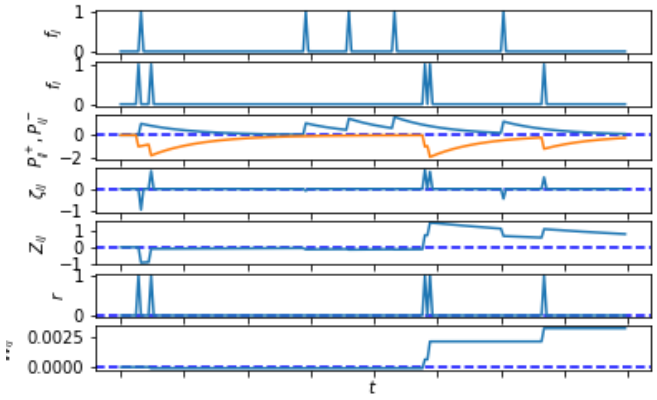


Figure 6.1: Learning process with MSTDPET

With the baseline learning rule, 50 experiments were performed. 32 experiments showed steady learning curves, 20 experiments obtained the best accuracy of 75% or above, 14 experiments had a bell-shaped learning curve and 4 failed to learn at all. The results of trials with good performance are comparable to those learned with the Urbanczik rule in Jordan's work. The average accuracy over epoch and an example of one learning curve is plotted in Figure 6.4.

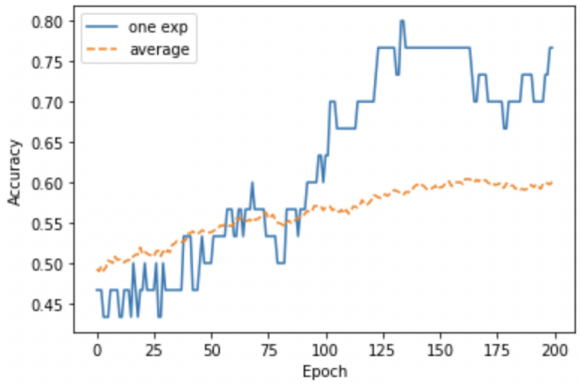


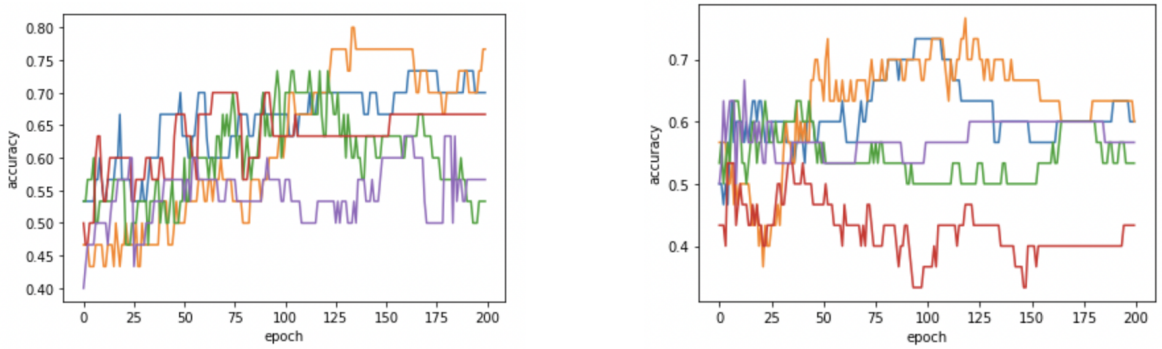
Figure 6.2: Learning curves of random pattern classification tasks with MSTDPET.

6.1.2 Evolutionary Search

After the task was successfully learned with the baseline learning rule, the evolutionary search was performed for evolving learning rules that could learn the task with comparable or even better performance than the baseline learning rule. As mentioned in section 5.2, the learning rate was fixed during the evolution. It was set to be the same as the learning rate tuned in the experiments with the baseline learning rule. Although MSTDPET was applied as the baseline learning rule instead of the Urbanczik rule, the weight update can also be considered as a function depending on synapse trace and reward. Thus, the input operands to the CGP were also the trace and reward. Since the weight update is now performed at each time step, the algorithm described in section 5.2 is updated as follows:

```
Initialise parents candidate solutions and their offspring
loop for each individual
    while number of trials not met do
        Initialise weights
        loop for each epoch
            loop for each pattern
                Reset neuron states and synapse trace
                loop for each time step
                    Feed input to the SNN
                    Update neuron states and synapse trace
                    Compute reward
                    Update weights with the candidate solution
                end loop
            end loop
        end loop
    end while
    Compute fitness
end loop
Select new parents and breed offspring for the next generation
```

A total of 10 runs of the evolutionary search were performed. Similar to the findings of Jordan, half of the experiments evolved the baseline learning rule. Some of the trials also evolved other learning rules with comparable performance. In addition to learning rules with a form approximately equal to the baseline rule such as $ER + constant$. Interestingly, some evolved learning rules that achieved good performance have the form of $E + constant$. Figure 6.3 shows the learning curves obtained by two examples of the evolved learning rules in the test experiments. Both learning rules have some trials with good performance and some with bad performances, while the learning rule with the term of $E \cdot R$ shows more stable learning behaviour. However, it was questioned that since the spike patterns and labels were both random, the evolved learning rules may not be plausible. Thus, the pattern classification task was modified in the following experiments.



(a) $\Delta W = ER + 0.5$ **(b)** $\Delta W = E + 0.5$

Figure 6.3: Learning curves of random pattern classification tasks.

6.2 Temporally Ordered Pattern Classification

6.2.1 Baseline Learning Rule

Since the pattern classification task was similar to the first experiment, the same baseline learning rule was implemented, which was the MSTDPET derived by Florian. The reward was defined the same as the previous task: if the label of the pattern is 1, the network receives a reward of 1 for each output spike. If the label of the pattern is 0, the network receives a reward of -1 for each output spike. 50 trials of the experiments were performed. It obtained stable learning curves in most trials and accuracy higher than 80 % in around half of the trials. The figure below plots the average accuracy with epochs of the 50 trials and the learning curve of one example trial. Similar to the previous experiment, it shows that the performance of the task improves over epochs in general.

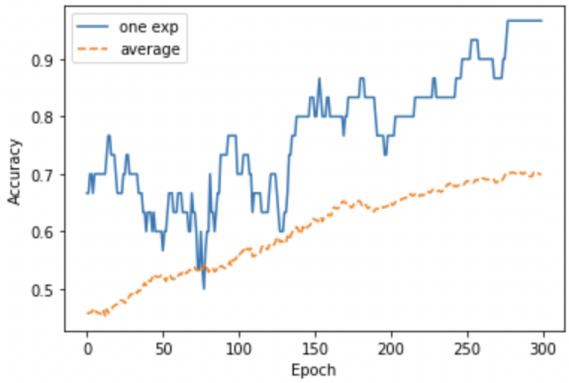
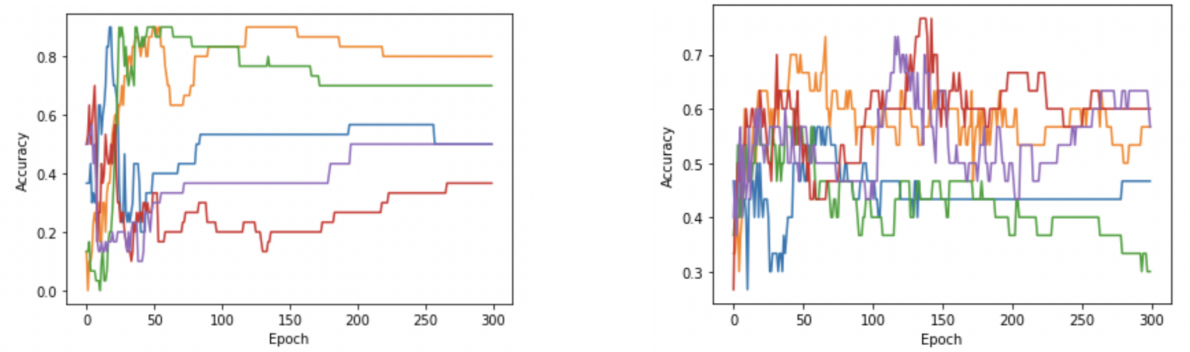


Figure 6.4: Learning curves of temporal ordered pattern classification tasks with MSTDPET.

6.2.2 Evolutionary Search

After the effectiveness of the MSTDPET on learning temporal ordered pattern classification task had been proven, the experiments of evolving new learning rules with CGP was performed. Similar to the evolutionary search in the previous task, the term $E \cdot R$ in the weight update function was replaced by the learning rules generated by CGP. Similarly, 10 runs of the evolutionary search were performed. Half of the experiments successfully evolved the baseline MSTDPET learning rule. Similar to the results of Jordan's work and the first preliminary experiments, the CGP also learned other learning rules which contain the term of $E \cdot R$. Some examples are: $ER + R, ER – R, ER + constant$..... These learning rules can learn the task by strengthening the connection pattern which responds correctly in case the additional term is less effective than the term of $E \cdot R$. Interestingly, the experiment shows that the trace itself can also learn the task well. The learning curves of five random trials where the learning rule is E are shown in Figure 6.5a. Two of the trials achieved a stable accuracy of 70% and above, the other learned in the beginning but the performance decreases after a number of epochs. Figure 6.5b shows the learning curves of $E+1$, which also had three trials with relatively high accuracy. One reason why the trace itself can learn the task may be caused by the difference of the trace due to the spike order. Thus, the weights of the first input neuron and the last input neuron were set to be zero in an additional experiment. However, the performance was similar to that of the original experiment, which means the task was not solved by classifying the track of the pattern. The learning rule of E can also be interpreted as $E \cdot R$, where the reward R is always 1. As a result, the connection will always be strengthened for every time step. With some cases of initializations where the trace for the correct prediction is higher than that of the wrong prediction, the trace can result in a comparable performance as the baseline learning rule. However, the learning rule will fail to learn when the initialization is the opposite. If the initial trace for the wrong prediction is higher, the strengthening of the pattern will carry on in the opposite direction as the baseline learning rule. This discovery indicates that the task can be learned as long as the ΔW for patterns with label 1 is sufficiently larger than that of patterns with label 0.



(a) $\Delta W = E$ **(b)** $\Delta W = E + 1$

Figure 6.5: Learning curves of temporally ordered pattern classification tasks trained with learning rules evolved.

6.3 XOR Pattern Classification without Prior Knowledge

After the above two pattern classification tasks have been successfully solved by baseline learning rules and evolutionary search has evolved learning rules with comparable performances, another XOR pattern classification task with the same complexity level was performed. As mentioned before, the goal of this experiment was to investigate the robustness of the evolutionary algorithm in case that the model had not been trained with prior knowledge. The evolutionary search was implemented directly on the SNN. The definition of reward was the same as the previous tasks and the reward and weights were updated at each time step.

Similar to the evolutionary search in the previous experiments, 10 search trials were performed and each trial has 100 generations. It was found that none of the experiments evolved learning rules with satisfactory performance. The output of the evolution for almost all the trials achieved a fitness of around 50%, which means the SNNs were performing random guesses instead of learning to classify. The learning rules evolved were also implausible and random. Most learning rules evolved have the form of $\Delta W = constant$. Learning curves of five random trials of an example learning rule are shown in Figure 6.6. It indicates that the evolved learning rule has little influence on training the model to perform the task. The reason for the failure may be due to the bad initialization of the hyper-parameters. In the previous experiments, hyper-parameters were tuned during the training process with the baseline learning rule. However, this process was skipped in this experiment and the hyper-parameters were not included in the evolutionary search. Thus, it was unknown whether the hyper-parameters were properly set or not. This issue will be further discussed in the next chapter.

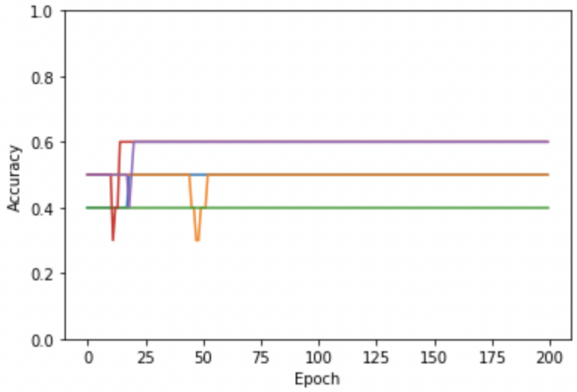


Figure 6.6: Learning curves of XOR pattern classification tasks with a learning rule evolved: $\Delta W = 1$.

7

Discussion of Preliminary Experiments

After the experiments of three tasks have been carried out, a discussion of the results presented in chapter 6 will be performed in this chapter. Since the preliminary experiments serve as a practice for understanding the research topic and an early attempt to reach the research objective, the discussion will focus on the research questions as listed in chapter 1. As mentioned in chapter 1, to answer the main research question of "how can the learning rules be evolved for training a predefined spiking neural network on continuous control problems", the expression method of the learning rules and the information needed for the evolutionary search should be figured out first. Thus, the discussion in section 7.1 will be about the expression method for the candidate learning rules. Next, the knowledge related to the evolutionary search obtained in preliminary experiments will be summarized in section 7.2. Finally, indications and inspirations for the later experiments will be concluded in section 7.3.

7.1 Expression Method

Since the goal of this work is to evolve learning rules for training SNNs, how to express the candidates of the learning rule should be figured out first. As mentioned in chapter 4, the approach was selected to be evolving the symbolic expression of the entire learning rule and CGP was utilized as the symbolic expression method. According to the results of the experiments on the random pattern classification and temporal ordered pattern classification, the selected expression method can be considered to be a good approach. Both these two experiments successfully evolved the baseline learning rules and alternative learning rules with comparable performances. Some of the alternative learning rules had structures different from that of the baseline learning rule. It reveals the advantage of evolving the entire symbolic expression over only evolving the coefficients of the Hebbian learning rule. Since the structure of the learning rule is not fixed, this approach allows larger search space and therefore has the ability to generate new plasticity rules with unconventional structures. Additionally, the learning rules with good performances discovered by the evolutionary search of both the experiments on the random pattern classification and temporal ordered pattern classification contain the term $E \times R$. The discovery indicated that this is the essential term for the plasticity learning rules and the rules can still learn tasks well with a bias to trace or reward. It demonstrates another advantage of the symbolic expression method over the ANN approach. Since the output of the evolutionary search is symbolic mathematical expressions, the evolved learning rules are interpretable, which is favorable for understanding the learning process of SNNs.

In the first two preliminary experiments, the CGP method demonstrated sufficient expression capability. Since the baseline learning rule has a simple expression with only one term, CGP can narrow down the search space by adjusting the number of columns of the index graph, which determines the maximum number of terms for the evolved learning rules. Thus, it could evolve the baseline learning rule rapidly. Additionally, by limiting the number of terms, it also eliminates solutions with extremely long expressions which are difficult to interpret. Moreover, the outputs of CGP are syntactic valid. Thus, no additional valid check program was needed.

7.2 Architecture of the Evolutionary Search

In the preliminary experiments, evolutionary searches both with or without prior knowledge were performed. In the experiments with prior knowledge, the SNNs were first trained with the baseline learning rule. During this pre-training, the appropriate learning rule, learning scheme, and hyper-parameters such as neuron characteristics and learning rate were derived and tuned. As described in section 6.1, the selection of the baseline learning rule and learning scheme is essential for the training performance. In the first preliminary experiment, the Urbanczik rule with the reward updated at the end of each episode failed to learn the task, while the MSTDPET rule with the reward updated at each time step learned the task well. Thus, the same learning scheme was applied in the evolutionary search, where the reward and weights were updated at each time step and the reward was defined the same as the pre-training. By doing so, it ensures that the baseline learning rule can be evolved from the evolutionary search. Additionally, it was found that tuning the hyper-parameters was also essential for successful learning. With a set of badly assigned hyper-parameters, the SNN can be either too active or inactive. In the former case, the neuron spikes almost at each time step. In the latter case, the neuron releases almost no spikes. Both cases make learning difficult. Thus, the hyper-parameters were first manually tuned in the pre-training and the same values were used in the evolutionary search.

Since the learning performance highly relies on the learning scheme and hyper-parameters, it was not surprising that the evolutionary search without prior knowledge failed to evolve effective learning rules. Since the evolutionary search for the last experiment was performed directly without the pre-training, the hyper-parameters were set to be commonly used values, and the trace and reward were defined using the general definition. Although more input variables such as traces with different decays were available for the evolution, it was still difficult for the evolution since it was unknown whether the search space includes a solution or not. Furthermore, even though there is a good solution, it may not be discovered due to the inappropriate hyper-parameters or learning scheme. The problem may be solved by including the hyper-parameters into the gene for evolution. However, it will largely increase the search space and will require much more computational power.

7.3 Later Experiments

It can be concluded from the discussion above that the symbolic approach with CGP can be considered as an appropriate expression method of the learning rules. Thus, the same expression method will be used in the later experiments. The experiments also indicated the importance of prior knowledge. Thus, in the next stage, the XOR pattern recognition task will be performed again by including prior knowledge in the evolutionary search. Similar to the previous experiments, the results of the evolution will be analyzed. The importance of prior knowledge will be further investigated by evaluating the operands involved in the evolved learning rules. Next, the main research question will be answered. The evolutionary search for effective learning rules for training SNNs on continuous control tasks will be performed. The research plan for the next phase is summarized in Figure 7.1.

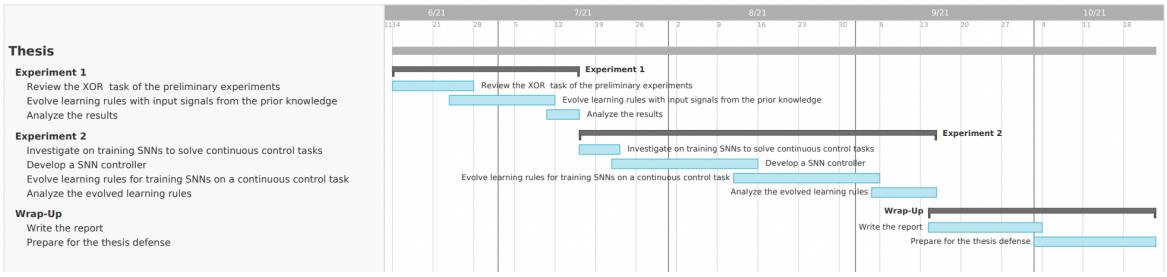


Figure 7.1: Research plan.

IV

Appendices

A

Simulation Configurations

Table A.1: Hyperparameter values for the three preliminary experiments.

Hyper-parameters	**Unit**	**Task 1**	**Task 2**	**Task 3**
N	-	50	50	2
M	-	30	30	10
T	ms	500	500	100
τ_I	ms	2	2	2
τ_V	ms	10	10	10
τ_e	ms	100	100	50, 100
θ	mV	1	1	1
η	-	0.001	0.001	0.001
μ	-	2	2	2
λ	-	4	4	4
epoch	-	200	300	200

Bibliography

Abusnaina, A. & Abdullah, R. (2014). Spiking neuron models: a review. *International Journal of Digital Content Technology and its Applications.*

Alber, M., Bello, I., Zoph, B., Kindermans, P., Ramachandran, P. & Le, Q. (2018). Backprop evolution. *arXiv preprint.* https://doi.org/arXiv:10808.02822

Alstrøm, P. & Stassinopoulos, D. (1995). Versatility and adaptive performance. *Physical Review*, *51*(5), 5027–5032. https://doi.org/10.1103/physreve.51.5027.

Andrychowicz, M., Denil, M., Colmenarejo, S., Hoffman, M., Pfau, D., Schaul, T., Shillingford, B. & de Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. *Advances in Neural Information Processing Systems*, *29*, 3988–3996.

Barto, A. G. (1985). Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, *4*(4), 229–256.

Bello, I., Zoph, B., Vasudevan, V. & Le, Q. (2017). Neural optimizer search with reinforcement learning. *arXiv preprint.* https://doi.org/arXiv:1709.07417

Bengio, Y., Lee, D., Bornschein, J. & Lin, Z. (2015). Towards biologically plausible deep learning. *arXiv prePrint.* https://doi.org/arXiv:1502.04156v2

Bing, Z., Meschede, C., Huang, K., Chen, G., Rohrbein, F., Akl, M. & Knoll, A. (2018). End to end learning of spiking neural network based on R-STDP for a lane keeping vehicle. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 1–8. https://doi.org/10.1109/ICRA.2018.8460482

Bohnstingl, T., Scherr, F. & Pehle, C. (2019). Neuromorphic hardware learns to learn. *Frontiers in neuroscience.*

Bohte, S. M., Kok, J. N. & La Poutré, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, *48*(1), 17–37. https://doi.org/10.1016/S0925-2312(01)00658-0

Booij, O. & Nguyen, T. H. (2005). A gradient descent rule for spiking neurons emitting multiple spikes. *Information Processing Letters*, *95*(6), 552–558. https://doi.org/https://doi.org/10.1016/j.ipl.2005.05.023

Brunel N., v. R. M. (1907). From frogs to integrate-and-fire. *Biological Cybernetics*, *97*(5), 337–339. https://doi.org/10.1007/s00422-007-0190-0

Chung, S. & Kozma, R. (2020). Reinforcement learning with feedback-modulated td-stdp. *arXiv preprint.* https://doi.org/arXiv:2008.13044

Davies, M., Srinivasa, N. & Lin, T.-H. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro*, *38*(1), 82–99. https://doi.org/10.1109/MM.2018.112130359

Diehl, P. U. & Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, *9*. https://doi.org/10.3389/fncom.2015.00099

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I. & Abbeel, P. (2017). Rl2: fast reinforcement learning via slow reinforcement learning. *ICLR.* https://doi.org/arXiv:1611.02779

Fang, H., Luo, J. & Wang, F. (2012). Fast learning in spiking neural networks by learning rate adaptation. *Chinese Journal of Chemical Engineering*, *20*(6), 1219–1224. https://doi.org/https://doi.org/10.1016/S1004-9541(12)60611-9

Finn, C. & Levine, S. (2018). Meta-learning and universality: deep representations and gradient descent can approximate any learning algorithm. *International Conference on Learning Representations*.

Finn, C., Xu, K. & Levine, S. (2018). Probabilistic model-agnostic meta-learning. *Advances in Neural Information Processing Systems*, *31*, 9516–9527.

FitzHugh, R. (1961). Impulses and physiological states in theoretical models of nerve membrane. *Biophysical Journal*, *1*(6), 445–466. https://doi.org/https://doi.org/10.1016/S0006-3495(61)86902-6

Florian, R. V. (2007). Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation*, *19*(6), 1468–1502. https://doi.org/10.1162/neco.2007.19.6.1468

Frémaux, N., Sprekeler, H. & Gerstner, W. (2013). Reinforcement learning using a continuous time actor-critic framework with spiking neurons. *PLOS Computational Biology*, *9*(4), e1003024. https://doi.org/10.1371/journal.pcbi.1003024

Gerstner, W., Kempter, R., van Hemmen, J. & Wagner, H. (1997). *A developmental learning rule for coincidence tuning in the barn owl auditory system*. Springer, Boston, MA.

Gerstner, W., Kistler, W. M., Naud, R. & Paninski, L. (2014). *Neuronal dynamics*. Cambridge University Press.

Ghosh-Dastidar, S. & Adeli, H. (2009). A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. *Neural Networks*, *22*(10), 1419–1431. https://doi.org/https://doi.org/10.1016/j.neunet.2009.04.003

Guo, Y., Wu, H., Gao, B. & Qian, H. (2019). Unsupervised learning on resistive memory array based spiking neural networ. *Neuromorphic Engineering*. https://doi.org/https://doi.org/10.3389/fnins.2019.00812

Hagenaars, J., Paredes-Vallés, F., Bohté, S. & de Croon, G. (2020). Evolved neuromorphic control for high speed divergence-based landings of mavs. *IEEE Robotics and Automation Letters*, *5*(4). https://doi.org/10.1109/LRA.2020.3012129

Hebb, D. (1949). *The organization of behavior*. John Wiley.

Hodgkin, A. & Huxley, A. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerves. *The Journal of Physiology*, *117*(4), 500–544. https://doi.org/10.1113/jphysiol.1952.sp004764

Hu, J., Niu, H., Carrasco, B., J.and Lennox & Arvin, F. (2020). Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, *69*(12), 14413–14423. https://doi.org/10.1109/TVT.2020.3034800

Huisman, M., van Rijn, J. & Plaat, A. (2021). A survey of deep meta-learning. *AI Review (AIRE) Journal*. https://doi.org/10.1007/s10462-021-10004-4

Hüttenrauch, M., Šošić, A. & Neumann, G. (2019). Deep reinforcement learning for swarm systems. *Journal of Machine Learning Research*, *20*(54), 1–31. https://doi.org/arXiv:1807.06613

Izhikevich, E. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, *14*(6), 1569–1572. https://doi.org/10.1109/TNN.2003.820440

Jordan, J., Schmidt, M., Senn, W. & Petrovici, M. (2021). Evolving to learn: discovering interpretable plasticity rules for spiking networks. *arXiv preprint*. https://doi.org/arXiv:2005.14149

Kistler, W. M. & Hemmen, J. L. v. (2019). Modeling synaptic plasticity in conjunction with the timing of pre-and postsynaptic action potentials. *Neural Comput*, *12*, 385–405. https://doi.org/10.1162/089976600300015844

Koch, G. (2015). Siamese neural networks for one-shot image recognition.

Lee, J., Delbruck, T. & Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience*. https://doi.org/https://doi.org/10.3389/fnins.2016.00508

Lobo, J., Laña, I., Ser, J., Bilbao, M. & Kasabov, N. (2018). Evolving spiking neural networks for online learning over drifting data streams. *Neural Networks*, *108*, 1–19. https://doi.org/https://doi.org/10.1016/j.neunet.2018.07.014

Lømo, T. (1966). Frequency potentiation of excitatory synaptic activity in the dentate area of the hippocampal formation. *Acta Physiol. Scand.*, *68*(277), 128.

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, *10*(9), 1659–1671. https://doi.org/10.1016/S0893-6080(97)00011-7

Markram, H., Lübke, J., Frotscher, M. & Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps. *Science*, *275*(5297), 213–215. https://doi.org/10.1126/science.275.5297.213.

Mead, C. (1990). Neuromorphic electronic systems. *Proceedings of the IEEE*, *78*(10), 1629–1636. https://doi.org/10.1109/5.58356

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., Jackson, B. L., Imam, N., Guo, C., Nakamura, Y., Brezzo, B., Vo, I., Esser, S. K., Appuswamy, R., Taba, B., Amir, A., Flickner, M. D., Risk, W. P., Manohar, R. & Modha, D. S. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, *345*(6197), 668–673. https://doi.org/10.1126/science.1254642

Mishra, N., Rohaninejad, M., Chen, X. & Abbeel, P. (2018). A simple neural attentive meta-learner. *In: International Conference on Learning Representation*.

Modha, D. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, *345*(6197), 668–673. https://doi.org/10.1126/science.1254642

Mohemmed, A. & Schliebs, S. (2012). Span: spike pattern association neuron for learning spatio-temporal spike patterns. *International Journal of Neural Systems*, *22*. https://doi.org/10.1142/S0129065712500128

Morrison, A., Diesmann, M. & Gerstner, W. (2008). Phenomenological models of synaptic plasticity based on spike timing. *Biological Cybernetics*, *98*(6), 459–478. https://doi.org/https://doi.org/10.1007/s00422-008-0233-1

Morrison, A., Aertsen, A. & Diesmann, M. (2007). Spike-timing-dependent plasticity in balanced random networks. *Neural Computation*, *19*(6), 1437–1467. https://doi.org/10.1162/neco.2007.19.6.1437

Munkhdalai, T. & Yu, H. (2017). Meta networks. *In: Proceedings of the 34th International Conference on Machine Learning*, 2554–2563.

Neftci, E., Mostafa, H. & Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, *36*(6), 51–63. https://doi.org/10.1109/MSP.2019.2931595

Niv, Y., Joel, D., Meillijson, I. & Ruppin, E. (2002). Evolution of reinforcement learning in uncertain environment: a simple explanation for complex foraging behaviours. *Adaptive Behaviour*, *10*(1), 5–24.

Oltean, M., Grosan, C., Diosan, L. & Mihaila, C. (2009). Genetic programming with linear representation: a survey. *International Journal on Artificial Intelligence Tools*, *18*(2), 197–238. https://doi.org/https://doi.org/10.1142/S0218213009000111

Orchard, J. & Wang, L. (2016). The evolution of a generalized neural learning rule. *Neural Networks*.

Paugam-Moisy, S., H.and Bohte. (2012). Computing with spiking neuron networks. *Handbook of Natural Computing*, 335–376. https://doi.org/https://doi.org/10.1007/978-3-540-92910-9_10

Pérez, J., Cabrera, J. A., Castillo, J. J. & Velasco, J. M. (2018). Bio-inspired spiking neural network for nonlinear systems control. *Neural Networks*, *104*, 15–25. https://doi.org/10.1016/j.neunet.2018.04.002

Pfister, J.-P., Toyoizumi, T., Barber, D. & Gerstner, W. (2006). Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning. *Neural Computation*, *18*(6), 1318–1348. https://doi.org/10.1162/neco.2006.18.6.1318

Poon, C. & Zhou, K. (2011). Neuromorphic silicon neurons and large-scale neural networks: challenges and opportunities. *Frontiers in neuroscience*, *5*(108). https://doi.org/https://doi.org/10.3389/fnins.2011.00108

Qiu, H., Garratt, M., Howard, D. & Anavatti, S. (2018). Evolving spiking neural networks for nonlinear control problems. *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, 1367–1373. https://doi.org/10.1109/SSCI.2018.8628848

Ravi, S. & Larochelle, H. (2017). Optimization as a model for few-shot learning. *International Conference on Learning Representations*.

Risi, S. & Stanley, K. (2010). Indirectly encoding neural plasticity as a pattern of local rules. *International Conference on Simulation of Adaptive Behaviour*.

Risi, S. & Togelius, J. (2017). Neuroevolution in games: state of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, *9*, 25–41. https://doi.org/arXiv:1410.7326

Ruf, B. & Schmitt, M. (1997). Learning temporally encoded patterns in networks of spiking neurons. *Neural Processing Letters*, *5*(1), 9–18. https://doi.org/https://doi.org/10.1023/A:1009697008681

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D. & Lillicrap, T. (2016). Meta-learning with memory-augmented neural networks. *Proceedings*.

Schmidhuber, J. (1987). Evolutionary principles in self-referential learning. *Diploma Thesis, Tech. Univ. Munich.*

Schrauwen, B. & Van Campenhout, J. (2004). Extending spikeprop. *Conference: Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference*, *1*, 471–475. https://doi.org/110.1109/IJCNN.2004.1379954

Seung, H. S. (2003). Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. *Neuron*, *40*(6), 1063–1073. https://doi.org/10.1016/S0896-6273(03)00761-X

Shim, M. S. & Li, P. (2017). Biologically inspired reinforcement learning for mobile robot collision avoidance. *2017 International Joint Conference on Neural Networks (IJCNN)*, 3098–3105. https://doi.org/10.1109/IJCNN.2017.7966242

Song, S., Miller, K. D. & Abott, L. F. (2000). Competitive hebbian learning through spike timing-dependent plasticity. *Nature Neuroscience*, *3*(9), 919–26. https://doi.org/10.1038/78829

Soula, H., Alwan, A. & Beslon, G. (2005). Learning at the edge of chaos: temporal coupling of spiking neuron controller of autonomous robotic. *In Proceedings of AAAI Spring Symposia on Developmental Robotics.*

Stanley, K. O. & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, *10*(2), 99–127. https://doi.org/10.1162/106365602320169811

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T. & Maida, A. (2019). Deep learning in spiking neural networks. *Neural Networks*, *111*, 47–63. https://doi.org/10.1016/j.neunet.2018.12.002

Toenlli, P. & Mouret, J. (2011). On the relationships between synaptic and generative systems. *13th annual conference on genetic and evolutionary computation*.

Urbanczik, R. & Senn, W. (2009). Reinforcement learning in populations of spiking neurons. *Nature Neuroscience*, *12*(3), 250–252. https://doi.org/10.1038/nn.2264

van Otterlo, M. & Wiering, M. (2012). Reinforcement learning and markov decision processes. *Reinforcement Learning. Adaptation, Learning, and Optimization*, *12*, 3–42. https://doi.org/10.1007/978-3-642-27645-3_1

Vikhar, P. A. (2016). Evolutionary algorithms: a critical review and its future prospects. *Proceedings of the 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, 261–265. https://doi.org/10.1109/ICGTSPICC.2016.7955308

Wang, J., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J., Munos, R., Blundell, C., Kumaran, D & Botvinick, M. (2016). Learning to reinforcement learn. *arXiv preprint*. https://doi.org/arXiv:161105763

Weng, L. (2018). Meta-learning: learning to learn fast. *lilianweng.github.io/lil-log*. http://lilianweng.github.io/lil-log/2018/11/29/meta-learning.html

Xie, X. & Seung, H. S. (2004). Learning in neural networks by reinforcement of irregular spiking. *Physical Review E*, *69*(4), 041909. https://doi.org/10.1103/PhysRevE.69.041909

Yuxi, L. (2017). Deep reinforcement learning: an overview. *arXiv PrePrint*. https://doi.org/arXiv:1701.07274