Indirect Training Algorithms for Spiking Neural Networks based on Spiking Timing Dependent Plasticity and Their Applications

by

Xu Zhang

Department of Mechanical Engineering and Materials Science
Duke University

Date: _____

Approved:

_____
Craig S. Henriquez, Supervisor

_____
Brian P. Mann

_____
Michael M. Zavlanos

_____
Marc A. Sommer

Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Mechanical Engineering and Materials Science
in the Graduate School of Duke University
2016

Abstract

Indirect Training Algorithms for Spiking Neural Networks based on Spiking Timing Dependent Plasticity and Their Applications

by

Xu Zhang

Department of Mechanical Engineering and Materials Science
Duke University

Date: _____

Approved:

_____
Craig S. Henriquez, Supervisor

_____
Brian P. Mann

_____
Michael M. Zavlanos

_____
Marc A. Sommer

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Mechanical Engineering and Materials Science
in the Graduate School of Duke University
2016

Abstract

Spiking neural networks have been used to investigate the mechanisms of processing in biological neural circuits or to propose hypotheses that can be tested in experiments. Because of their biological plausibility and event-based information transmission, Spiking Neural Networks (SNNs) have suggested as alternatives to Artificial Neural Networks for pattern recognition, classification and function approximation problems with fewer neurons. In machine learning, SNNs has been shown to be able to solve pattern and robotic control. For SNNs to be used for such problems, they must incorporate some mechanism for learning. Current methods to train SNNs use learning algorithms which adjust the synaptic weights according to an update rule. In most cases the weights are modified directly. In potential applications such as driving plasticity in neural culture (in-vitro) and training neuromorphic chips the directly manipulation of synaptic weights is not possible. Therefore, indirect algorithms, which cause the SNNs to learn based on some biological learning mechanisms using stimulation of neurons offer significant advantages over the existing algorithm for these real world applications.

Indirect algorithms train the neural network by using external stimuli to modulate the synaptic strengths of a neural network according to synapses intrinsic mechanisms for plasticity. The training algorithms have been demonstrated in both Integrate and Fire neurons and more biologically realistic neural networks. In this thesis, four indirect methods to drive the synaptic weights to its desired value in a network through

Spike Time Dependent Plasticity (STDP) are developed: Indirect Perturbation, Indirect Stochastic Gradient, Indirect ReSuMe, and Indirect Training with Supervised Teaching Signals. These algorithms are used to solve the temporal and spatial input-output mapping problem using temporal coding. The other type of problem is to mapping input output firing rates using rate coding.

To test the algorithms, SNNs are used to control both virtual and real world robots. For the real world robots with SNNs, known and Neurorobots, two types of robot localization techniques are used: *Optitrack*, using ceiling mounted cameras and onboard markers, and embedded cameras. Both small and large SNNs with biologically realistic neurons are used to drive the neurorobots are modeled with input coming from *Optitrack* or the cameras with GPU accelerated SNN simulator. The results show that the indirect perturbation and indirect stochastic gradient algorithms can train an SNN to control the robot to find targets and avoid obstacles even in the presence of sensor noise. The results also show that indirect training with supervised training signals algorithm can train a feedforward network with 1000s of neurons to process and output the correct movement commands to localize a target using from real time images captured from an embedded camera. Finally. an indirect version of the Remote Supervised Method (ReSuMe) algorithm was developed using a more biologically realistic form of Spike-Timing Dependent Plasticity to produce a specific temporal pattern of spiking from a group of neurons. The indirect algorithms developed in this thesis may eventually allow the ability to train in vitro and in vivo biological circuits to perform specific tasks using patterns of electrical or light stimulation.

I dedicate this thesis to God for His blessing, to my wife for her consistent love, to the MEMS department and Graduate School for the funding support and valuable suggestions.

Contents

List of Tables

List of Figures

List of Abbreviations and Symbols

Symbols

Below are the descriptions of a list of symbols used in this dissertation.

I_{syn}	Synaptic current.
I_{Inj}	External inject current.
I	Total current given to a neuron.
I_R	Current that goes through resistance.
I_C	Current that charge the capacitance.
R	Membrane resistance.
C	Membrane capacitance.
C_m	Membrane capacitance.
$\delta(\cdot)$	Delta pulse.
$\alpha(\cdot)$	Alpha pulse.
V_{th}	Threshold.
τ_m	Membrane potential time constant.
E_L	Resting potential.
Δ_Q, Δ_E, V_T	Constant voltages.
v	Membrane potential.
u	Membrane recovery variable.
a, b, c, d	Dimensionless parameters for Izhikevich model.

t_f	Firing time.
$g_{syn}(t)$	Synaptic conductance.
$\bar{g}_{syn}$	Synaptic conductance amplitude.
f	Normalization factor for synaptic conductance.
$\tau_{decay}, \tau_{rise}$	Time constants for the decaying speed and rising speed of conductance.
t_0	Rising time of the synaptic conductance.
t_{peak}	Peak time of the synaptic conductance.
h	Temporary variable for conductance model.
W	Synaptic weight.
A_+, A_-	Maximum increase and decrease of weight per spike pair.
Δt	Temporal difference between spikes.
τ_+, τ_-	Time constants for STDP
t_{pre}, t_{post}	Firing times of pre and postsynaptic neuron.
x_i, y_i	Local variable for implement STDP.
t_i^f, t_j^f	Firing time of pre, postsynaptic neurons.
τ_x, τ_y	Time constants for implement STDP.
w_{ij}	Weight between neuron i and j.
$t_{i,k}$	k_{th} Firing of neuron i.
c_k	k_{th} Square pulse given to the neuron.
β	Width of the square pulse.
Δ^{abs}	Time length of refractory period.
T	Time costs for a neuron to fire given constant input.
ω	Height of the constant input.
R_m	Membrane resistance.
f_L^*, f_R^*	Desired average firing frequencies of left and right motor.

h_L, g_L	Left terrain and target sensor values.
h_R, g_R	Right terrain and target sensor values.
x_L, y_L, x_R, y_R	Positions of the left and right target sensors.
d_L, d_R	Euclidean distance from left and right target sensor to the target.
$\lambda, \alpha, \gamma, \sigma$	Constant scalers.
r	Roughness of the terrain.
n	Number of possible pairs.
N	Number of neurons.
$e_{k,l}$	l_{th} training error of k^{th} epoch.
t_e	Time length of one test epoch.
$\mathbb{R}$	Membrane resistance.

Abbreviations

SNN	Spiking neural network.
STDP	Spike timing dependent plasticity
ReSuMe	Remote supervised method.
CMOS	Complementary metaloxidesemiconductor.
IF	Integrate-and-fire.
LIF	Leaky integrate-and-fire.
ISGD	Indirect stochastic gradient descent.
ANN	Artificial neural network.
MEA	Multi-electrode array.
HH	Hodgkin-Huxley.
CSIM	Circuit SIMulator.
CPG	Central pattern generator.
CH	Chattering.

IB	Intrinsically bursting.
FS	Fast spiking.
LTS	Low-threshold spiking.
RS	Regular spiking.
TC	Thalamo-cortical.
RZ	Resonator.
PSP	Postsynaptic potential.
LTP	Long-term potentiation.
LTD	Long-term depression.
CNN	Convolutional neural network.
SVM	Support vector machine.
ISI	Interspike intervals.
ROS	Robot operating systems.
CUDA	Compute unified device architecture.
FPGA	Field-programmable gate array

Acknowledgements

Many thanks to Mechanical Engineering and Materials Science department's consistent funding supports and National Science Foundation's ECCS Grant 0925407. Thanks to my adviser Dr. Henriquez, from whom I learned how to improve myself to be both a good researcher and a good programmer skilled in different programming languages that helped me find a perfect job related to Spiking Neural Networks. Thanks Dr. Mann's many suggestions during my transition and helps on my funding support. Thanks Graduate School on my tuition waive and communications during my difficult time. Thanks Dr. Zavlanos' permission so that I can do experiments using the robot lab's equipments. Thanks Dr. Sommer for his agreement to be my committee member and many valuable suggestions.

1

Introduction

Spiking neural networks are a more biologically realistic form of artificial neural networks. Information is transmitted in the network though brief events known as spikes, to mimic action potentials, and via time-dependent dynamics and delays, to represent electrical and chemical synapses in the brain. SNNs can represent information through both the spiking rate and the the relative timings of the spikes [99]. As a result, SNNs can replicate biological phenomenon observed in in vitro neuronal cultures or in vivo animal brains [170, 21] and can be used to produce hypotheses that can be tested in experiments or aid in interpreting results. Along with studying biology, spiking neural networks are beginning to be used for solving problems in pattern recognition, classification and function approximations [82, 24, 162, 57, 75, 28]. [101] has shown that SNNs can solve the same problem as ANNs by using fewer neurons.

Because of their potential advantages, SNNs have received increased attention as potentially powerful computational platforms that can be implemented in software or hardware. Maass introduced the concept of a liquid state machine with fading memory formed by a recurrent SNN for real-time computing on time varying input

streams for applications such as classification [98]. Software implementations of SNNs have also had a long history in neuroscience using both biophysically realistic and simplified models of neurons and synapses [19] Models have been constructed to investigate a wide range of phenomena such as the origin of brain oscillations [36] and central pattern generators underlying animal locomotion [86]. More recently, SNN have been used in the emerging field of computational psychiatry that seeks to link the emergent circuit dynamics to decision-making [117] and an actual behavior and in neurorobotics that seeks to integrate sensor data to drive robot movement or response [8].

Programmable hardware implementations of simplified models of neurons and synapses form the basis of neuromorphic chips [164]. Recently, researchers at the University of California, Santa Barbara, and Stony Brook University constructed a functional neuromorphic chip with hardware elements alone that mimic the digital neurons and analog synapses of a brain [130]. The key to this breakthrough was the incorporation of a two terminal device, known as a memristor, in which the resistance to current depends on the currents that have flowed through them in the past, to serve as the synapse. The development of neuromimetic devices may not only lead to new image and language processing technologies but may underlie new event-based sensory prosthesis to correct blindness or deafness. The primary challenge of these new computing platforms that use SNNs is that they cannot be programmed using traditional approaches. Like a real brain, the SNN implemented in software or on a neuromorphic chip needs to learn the desired behavior [164].

There is growing experimental evidence that learning in the brain involves changes in the strength of synaptic connections (plasticity) in the neuronal circuits [46]. One form of synaptic plasticity depends on the timing of the spikes in the neuron before the synapse (pre-synaptic) and the neuron after the synapse (postsynaptic). Markram et al. were the first to show that when a presynaptic spike is followed by

postsynaptic spike there is enhancement in the strength of the synapse and when a postsynaptic spike is followed by a presynaptic spike there is depression of the strength of the synapse [105]. Bi and Poo showed that the amount of strengthing or weakening depended on the timing of the pre and post-synaptic spike. This change in synaptic strength that depends on spike times is known as spike-timing dependent plasticity (STDP) [9].

Because of its biological basis and relative simplicity, STDP has been used in both software and hardware versions of SNNs. A number of supervised and reinforcement learning algorithms have been developed for training spiking neural networks to perform function approximation or classification by means of a learning rule that adapts the synaptic strengths via STDP plasticity. One of the first supervised learning algorithm for spiking neural networks, Spike-Prop, was inspired by Error Backpropagation algorithm for ANNs [136]. The Spike-Prop algorithm is based on a single-spiking structure, which means that the input, hidden, output layer neurons can only emit a single spike during each test. Additionally, Spike-Prop algorithm is subject to sudden jumps in the training error, called surges, which change the course of learning or even lead to a failure [157]. The use of adaptive learning rate has been found to reduce surges and variants of Spike-Prop have been developed that allow for multiple spikes in the hidden layers [146, 147]. ReSuMe is an example supervised learning algorithm that is based on the classical Widrow-Hoff rule which uses combination of the spike time-dependent plasticity (STDP) and anti-STDP learning windows to produce a desired output spike train in response to a spatiotemporal pattern [128]. The Chronotron algorithm is similar to ReSuMe in that it uses spike timing [51]. A trained network using this method has been shown to have a higher memory capacity than ReSuMe. Delay learning re-mote supervised method (DL-ReSuMe) combines the delay shift approach with ReSuMe to improve the learning performance and biological properties [155]. Recently, Multi-DL-ReSuMe has been

developed that uses multiple output neurons with each neuron classifying a single type of output [156]. Multiple Spike Pattern Association Neurons (SPANs) in a single layer network can be trained to solve classification problem with higher accuracy than a single SPAN [115]. In some algorithms, a global reward was used to simulate a neurochemical signal for modifying synaptic weights via STDP [138, 124, 91, 78, 81]. Spike-driven synaptic plasticity (SDSP) was used to train a spiking neural network to perform pattern recognition [82].

Unfortunately, most of the existing training algorithms rely on direct the manipulation the synaptic weights via a learning rule. Such direct manipulation is not possible in a biological circuit and may be limiting in certain implementations of neuromorphic chips using memristor technologies. However, it is possible to stimulate a subset of neurons either by electrical or light stimulus [174] to induce changes at the synapses. For example, light-sensitive neural systems grown in vitro can be stimulated to fire precisely in milliseconds scale [174]. Therefore, these indirect stimuli can be used as a teaching signal to force the neurons to fire in a specific pattern so that the synaptic weights can be indirectly modulated to train the SNNs.

In this thesis, I develop four indirect training algorithms for SNNs. Because SNNs can include both rate-coding or spike-coding [55], most existing learning algorithms have been designed specifically for solving problems limited to specific coding schemes [68]. The indirect training algorithms developed here are also tested on both rate coding and spike coding problems for wide applicability. For rate coding, the algorithms are used to train a large scale SNNs for controlling both virtual simulated neurorobot for finding a target while avoiding obstacles and an actual real-time neurorobot in an environment populated with obstacles for finding targets. For temporal coding, an SNN is trained to mapping randomly generated input spikes into desired firing times of the output neurons.

1.1 Training CMOS and Memristor Devices

The complementary metal-oxide-semiconductor (CMOS) has been used in microprocessors, microcontrollers, static RAM and other digital logic circuits since the early 1960s. CMOS has two important strengths including high noise immunity and low static power consumption [142]. Therefore, it does not produce as much wasted heat as other kinds of logic such as transistor-transistor logic (TTL) or NMOS logic, which usually have standing current even when not changing state. In comparison with electronic devices, the brain is extremely energy efficient using about 15 kilocalories per hour, which is the same amount of energy that a quad-core CPU uses in 20 minutes (around 20 watt hours). So the brain uses about 3 times less power to support 40,000 times more synapses than transistors. Therefore, that the brain is over 100,000 times more energy efficient. Nevertheless, CMOS neurons are being developed to implement neural networks in hardware VLSI chips [96, 95, 39, 66].

Static RAM has been designed to modeling the synapse's behavior on nanoscale chips [143]. Even though SRAM can provide fast access time and is stable, it always needs a voltage to be applied for retention [171]. The memristor, considered as the fourth fundamental circuit element, behaves very similarly to the biological synapse. It was discovered by Chua in 1971 and further developed by Williams at Hewlett Packard in 2008. The configurable resistance of the memristor maintains its state even when the power is disconnected. This allows the memristor to serve as a low-cost electrical element in the next generation of computers or chips [72, 13].

By combining CMOS and memristor technology, neuromorphic chips have been designed to solve some real world problems in both neuroscience and robotics [85, 152, 172, 172, 93]. Memristors can incorporate Spike-Timing-Dependent Plasticity as a learning mechanism [144, 131, 105]. Unfortunately, most STDP learning algorithms are unsupervised [108, 37, 107, 167]. Therefore, indirect training algorithms are

needed to enable supervised learning or reinforcement learning based on the STDP to greatly improve the potential application areas of neuromorphic chips.

1.2 Training Neuronal Cultures and Neurochips

Culture of neurons have been used to study processing in the brain and for studying animal learning, plasticity, and memory, [163]. The cultured neural networks are typically stimulated and recorded using an input/output electronic device called multi-electrode array (MEA), which make the two ways communication possible between researchers and neural networks. Using this framework, Potter et al. used cultured neurons to control simple robots termed hybrot or animat. By running hybrot in the real environment, they have been able to investigate the underlying learning and plasticity in a biologically realistic context [5, 35]. However, these cultures have proven extremely difficult to train. Traditional neural network learning algorithms that operate directly on the weights are not applicable to real neuronal cultures. As a result, these robots can only accomplish simple tasks like running in straight lines. In other words, there are currently no efficient methods to train the biological neural networks.

Although an MEA can both record and stimulate neuronal cultures, it does not providee high temporal and spatial precisions. Optical control of neurons has been proposed to overcome this limitation[173, 174]. Optogenetics combines genetics and optics for controlling well-defined events within specific cells of living tissue [114]. By using patterns light, it may be possible to achieve higher temporal-spatial resolution. In addition, optogenetics enable the ability to either hyperpolarize or depolarize cells using different wavelengths of light.

For implementing this technique *in-vitro*, randomly connected neural networks can be grown by the method called Banker Cultures [49], which grow neurons on top of a monolayer of astrocytes. By using the MEA, the firings of cultured out-

put neurons can be recorded and computed for real-time firing rates or integrated signals. In [16], Multi-Channel System recording hardware and software setups allow researchers to record the neuronal activity on millisecond timescales. Therefore, indirect training algorithms based on stimulating neural network using optogenetics and recording outputs using MEAs are needed to make solving reverse engineering problems using biologically neural culture possible.

Because of the small number of neurons, few connections and the ability for precise control compared with traditional neuronal cultures, the Neurochip could also benefit from indirect training algorithms. According to Pinelab at Caltech, the goal of the neurochip project is to design and fabricate a silicon-micromachined device that continuously records from and selectively stimulates individual neurons that are part of a small network of cultured neurons." Studies have been done on mapping connections between individual neurons grown in neurochip and analyzing learning [42, 103]. In one implementation of a Neurochip, 16 neurons on a 4*4 array of "neurocages" were grown. The Neurochip cultures show suprathreshold activity starting around 10-14 DIV, which is consistent with the time range for formation of functional synapses and network maturation in previous studies of hippocampal cultures. Whether the observed responses in neurochip are from mono- or polysynaptic connections are also investigated [42]. In this thesis, a biological plausible small network is modeled and trained by using indirect training methods, which are applicable to biological neurochip in future.

1.3 Spiking Neural Networks and Modeling Software

Neural network models can be classified into three generations: perceptrons, activation-function based networks, and spiking neural networks (SNNs). The first generation uses *McCulloch-Pitts neurons* as the computational elements, which are only known as perceptrons. Multilayer perceptrons, Hopfield nets, and Boltzmann machines

belong to this generation. These networks only generate digital output, the main weakness of these models. However, for computation with digital input and output, they are *universal*. To overcome the weakness of the first generation, the second generation of neural networks uses computational node models that include an "activation function". This type of neural network can have continuous input and output, where the input is the weighted sum of the outputs from the previous layer. The most commonly used activation functions include *sigmoid* function, binary step function, bipolar step function, sigmoidal function and ramp function. The second generation includes feedforward, recurrent neural networks, and radial basis function neuronal model. It has been shown in [102, 32] that the second generation uses fewer neurons to solve the same problem than the first generation. This generation can compute input-output mapping with analog signals. A number of gradient descent algorithms like backpropagation were designed for training this type of neural networks [137, 139, 181]. Even though ANNs have been shown capable of solving many classical machine learning and control problems such as pattern recognition, function approximation, and robotic control, they lack some of the basic characteristics of biological neural networks. First, these ANNs do not incorporate the time-based information transmission between neurons. Also, they do not have the ability to replicate some basic phenomena that commonly appear in biological neural networks like bursting and synchrony [104]. Thus, the third-generation neural network model, SNN, increases the level of biological realism by modeling neural spikes[45]. Neurons in the network communicate with each other via action potential spikes, often with the same time course. Therefore, the specific timings of those spikes matter more than shapes of individual spikes. This enables fast and energy saving transmission of information in the network.

Before picking a spiking neural network, we must evaluate the trade-off between the computational efficiency and biological realism. For example, Hodgkin-Huxley

type models for the spiking which incorporates ion flows and other dynamics, are computationally expensive and difficult to analyze, limiting networks to only a small group of neurons. While computationally challenging, these types of models can serve as an important reference for the derivation of other simple models. The four-dimensional Hodgkin Huxley model has been reduced to simpler two-dimensional models. Morris-Lecar model, Fitzhugh-Nagumo model, and Izhikevich model are several examples of the two dimensional models [118, 48, 79]. Further reduction is often needed for more analytical analysis of both learning and memory. Therefore, a reduction from two-dimensional to one-dimensional gives us one of the most popular neuron model called Leaky Integrate and Fire (LIF) neuron. The LIF neuron is a one-dimensional model described by a single Ordinary Differential Equation (ODE). It has been defined as a canonical model for spiking neurons because it can be simply studied analytically and also being sufficiently plausible to model several essential characteristics of a neural system. In this thesis, LIF neurons are used for deriving the analytical gradient algorithm; Izhikevich neurons are used to model SNNs in a GPU for controlling a mobile robot in real experiment environment and HH neurons are used for modeling a small neurochip.

There are many different SNN simulators with different level of computational speed and biological models. CSIM (Circuit SIMulator) can simulate heterogeneous networks with different neural and synapse models. It's written in C++ with a MEX interface to Matlab. The main strength of CSIM is that it has many different types of neuron and synapse models. The biggest weakness of CSIM is its simulation speed. Brian 2 [64], is another SNN simulator written in python and runs slightly faster than CSIM. A big advantage compared with CSIM is that users can define any type of neural and synapse models using ODES. CSIM users can only use pre-defined models. Neither, CSIM or Brian 2 can be used in a real-time environment. For simulating large neural network in real time or faster than real time, we used a

GPU accelerated simulator called CarlSim, which can simulate thousands of neurons faster than real time. This fast speed is really important for the use of SNNs in a physical robot which needs to interact with an obstacle or a moving target.

1.4 Application of SNNs in Neurorobotics

Neurorobotics is an emerging field that combines of neuroscience, robotics, and artificial intelligence. It has attracted many research groups to study neural systems and behavior by implementing simulated SNNs in a robot, which can interact with the real environment. In contrast to simulated environments, real environments are rich, complex and noisy. These real-world challenges require more sophisticated networks which are computationally expensive and difficult to train. Four characteristics of a neurorobot are: 1. robot is engaged in a behavioral task. 2. the robot is operating in a real environment. 3. the robot has interaction with the environment through sensory inputs. 4. The robot is controlled by a nervous system that simulates the brain's architecture at some level.

There are three main research directions of neurobotics: motor control, learning and value systems. Neurorobots are useful for studying animal locomotion and motor control to improve robotic controller designs. Central Pattern Generators (CPGs), networks of motorneurons that repeat a rhythmic firing pattern without extra stimuli, have been applied to robot locomotion [76, 84, 77, 41, 94]. Ijspeert's group designed an amphibious salamander-like robot that can swim and walk. This neurorobot was used to test whether their CPG model can reproduce the behavior of salamander locomotion both in water and on the ground. Endo's group at Tokyo Institute of Technology built the first successful biped locomotion system on a full-body hardware humanoid robot by using CPG to control the robot's legs.

Neurorobots are also being used to study motor control by developing predictive controllers that include more smooth, accurate movements. With training, the

robot can predict the environment and produce motor commands before the reflex response. Robots inspired by this have been developed and tested in obstacle distributed environment [112, 129].

A recent study in UC Irvine uses a cortical neural network model implemented in CarlSim running on a GPU to control an R/C car for reaching its target while avoiding obstacles [8]. The work mainly focuses on building primary visual cortex (V1), middle temporal area (MT) to help the robot process the information from an android phone camera and produce motor commands to steer the robot around obstacles towards its goal. Currently, there is no learning in these robots. The synaptic weights in the network are tuned offline by hand. Also, the computations are done on a desktop rather than directly on the robots.

In this thesis, I implement an SNN for the control of the neurobot. In contrast to previous studies, indirect training is used to obtain the synaptic weights. As noted earlier, the on advantage of indirect training algorithms is that they can be implemented in neuromorphic chips which may be used for on-board learning in mobile robots. The robots learn to avoid obstacles and to reach its target according to the biologically plausible STDP rule.

1.5 Training Algorithms for Spiking Neural Networks

The training algorithms for SNNs can be classified as either unsupervised learning or supervised learning. For unsupervised learning, the data provided to SNNs has no label and therefore no feedback about its performance is returned to the network. A normal task is to find and respond to data correlations in statistics. Hebbian learning and STDP are the classical examples [70, 108]. Beside finding the data correlations, unsupervised learning goal can also include data classification.

For supervised learning, neural network inputs and desired outputs are both required to evaluate and improve the performance of the neural network (mapping

between inputs and outputs). An error signal is defined by an error function to measure the difference between the actual output and the desired outputs. In supervised learning, the error signal is used to update the synaptic weights. Most gradient descent learning rules are supervised learning.

The current SNN training algorithms are still very limited compared with ANNs because the response of an SNN is not in closed-form and have to be solved numerically using a system of differential equations. In addition, the complex spike patterns have to be decoded into the lower-order continuous output for control or to evaluate the system-level performance.

Spike-Prop as one of the supervised learning algorithm for SNNs is inspired by Error BackProp algorithm for ANNs [14]. In Spike-Prop, each neuron is restricted to only one spike during a certain period. Similar restrictions also apply to extensions of this algorithm [15, 160], which limits the applications of this algorithm on most spike patterns. SpikeProp is only suitable to use 'time-to-first-spike' coding scheme. Also, this method is only defined for feedforward neural networks. [160] improved SpikeProp to recurrent SNNs even though it still requires one spike per neuron. Including momentum term in weight update equation significantly improves the convergence speed of SpikeProp [169]. Several other backprop methods have adapt to more diverse spike patterns so that it can include rate coding [135, 47, 150].

Belatreche et al. studied the applicability of evolutionary strategies (ES) on supervised learning of SNNs [6]. Compared to genetic algorithms, ESs' primary operator is mutation other than crossovers [62, 149]. In this algorithm, both the synaptic weights and synaptic delays are updated during learning. ESs are expected to converge to global optimal networks compared with gradient descent algorithms. One weakness of this algorithm is its computational complexity, which is much higher comparing with gradient-based algorithms. Also this method requires that the synaptic weights be set directly, which is not applicable to biological neural networks.

A number of experimental studies have shown that synaptic plasticity of biological neurons are driven by Hebbian learning mechanisms, such as STDP [105, 31, 92, 113, 165]. The synaptic weights change based on the temporal difference between the firing times of pre- and postsynaptic neurons. Supervised learning algorithms based on STDP have been tested on nonlinear function approximation and classification problems [126, 154]. STDP has also been used for reinforcement learning of SNNs for solving nonlinear function approximation and classifications [125, 90, 43, 52, 50]. The reinforcement signals simulate chemical rewards that modulate the STDP rule that controls the synaptic weight changes. Spike Driven Synaptic Plasticity (SDSP) has recently been applied on pattern recognition by assuming the synaptic weights are known [82].

Most of the current algorithms reviewed above have been used for SNNs performing function approximation or classification . However, none of them is applicable to biological neural networks or memristor-based computer chips because of their direct manipulation of synaptic weights. Therefore, indirect training algorithms have been proposed by stimulating training neurons during training periods to modulate the synaptic weights based on STDP learning rule. The methods described here have been both tested both in virtual environments and in real environments.

1.6 Discussion

New developments in neural stimulation and recording technologies are revolutionizing the field of neuroscience, giving scientists the ability to record and control the activity of individual neurons in the living animal's brains, with very high spatial and temporal resolution [34]. Otogenetics, which can be used to control cell firings in neurons *in-vitro* or *in-vivo*, is enabling the ability to determine the regions of the brain that are primarily responsible for encoding particular stimuli and behaviors. Despite this remarkable progress, the relationship between biophysical models of synaptic

plasticity and circuit-level learning, also known as functional plasticity, is poorly understood. This gap has recently been considered as an outstanding challenge in reverse engineering of the brain [158].

Many spiking neural network (SNN) learning algorithms have been developed to model and replicate both the synaptic plasticity and circuit-level learning mechanisms observed in biological neuronal networks [44, 59, 166, 175, 145, 153, 53]. Experiments have shown that learning in the brain is correlated to the change in synaptic efficacy or synaptic strength [38]. Developing learning rules for updating synaptic weights, has been the emphasis of both artificial neural networks (ANN) and SNN learning algorithms to date. In addition, SNN learning algorithms inspired by biological mechanisms, such as as spike-timing dependent plasticity (STDP) [105, 9, 120], have recently been roposed to modify synaptic weights according to a learning rule modeled based on STDP or Hebbian plasticity, to optimize the network performance [125, 90, 43, 52, 50, 82]. Other SNN learning algorithms include Spike-Prop [147, 58] and ReSuMe [128], which use classical backpropagation and Widrow-Hoff learning rules in combination with STDP to adapt the synaptic weights so as to produce a desired SNN response. When trained by these approaches, computational SNN have been shown to be very effective at solving decision and control problems in a number of applications, including delay learning, memory, and pattern classification [51, 155, 156, 116].

Despite their effectiveness, none of the computational SNN learning algorithms to date have been implemented or validated experimentally on biological neurons *in vitro* or *in vivo*. Such experimentation could help develop plausible models linking synaptic-level and functional-level plasticity in the brain, and also enable many potential neuroscience applications by closing the loop around the recording and the control of neuron firings. Existing SNN learning algorithms, however, are difficult to implement and test experimentally because they utilize learning rules that require

the direct manipulation of synaptic weights. Experimental methods for regulating synaptic strengths in biological neurons, for example via manipulating intracellular proteins or neurotransmitters such as AMPA receptors [40], do not lend themselves to the implementation of parallel and frequent weight changes, followed by the observation of network performance, as typically dictated by SNN learning algorithms.

To overcome this fundamental hurdle, I have developed four indirect SNN learning paradigm in which the learning rule regulates the spatiotemporal pattern of cell firings (or spike trains) to achieve a desired network-level response by indirectly modulating synaptic plasticity [179, 178]. Because these learning algorithms do not rely on manipulating synaptic strengths directly, they can theoretically be implemented experimentally by delivering the neural stimulation patterns determined by the algorithm to biological neurons using light or electrical stimulation. As a first step, this new indirect learning paradigm was demonstrated by showing that the synaptic strengths of a few neurons could be accurately controlled by optimizing a radial basis function (RBF) spike model using an analytical steepest-gradient descent method [178]. As a second step, the method was extended to networks with up to ten neurons by introducing an unconstrained numerical minimization algorithm for determining the centers of the RBF model, such that the timings of the cell firings could be optimized [179]. The latter approach was also shown effective at training memristor-based neuromorphic computer chips that aim to replicate the functionalities of biological circuitry [73, 110]. Because they are biologically inspired, these neuromorphic chips are characterized by STDP-like mechanisms that only adjust CMOS synaptic strengths by virtue of controllable applied voltages analogous to neuron firings. Therefore, they too are amenable to a learning paradigm that seeks to regulate the spatiotemporal pattern of cell firings (or spike trains) in lieu of the synaptic strengths.

This thesis presents four indirect training algorithms including indirect pertur-

bation algorithm, indirect stochastic gradient descent (SGD) algorithm, indirect ReSuMe algorithm and indirect training by supervised teaching signals algorithm. These algorithms are used to solve problems based on temporal coding and rate coding. More specifically, indirect perturbation algorithm and indirect SGD algorithm are used to train an unicycle modeled virtual insect for navigate in an unknown terrain while avoiding obstacles. Indirect perturbation algorithm is also used to train an Izhikevich neuron modeled neural network for controlling an *Optitrack* tracked robot to navigate in an indoor laboratory and find its target while avoiding obstacles. Indirect ReSuMe method is used to train an SNN for spatial and temporal mapping of the network input and output spikes. Indirect training by supervised teaching signals algorithm is finally described and used to train a large size SNN with thousands neurons to control a neurorobot with a embedded camera. Using the trained SNN, the robot can use the camera image to give the robot appropriate motor controls.

2

Spiking Neural Network Model

2.1 Integrate-and-Fire Model

The details of dimensional reduction of mathematical neural models are in [83]. Reduction from two-dimensional to Integrate-and-Fire model can be found in [1]. Compared to other neural models, Linear Integrate-and-Fire (LIF) neurons have the highest computational efficacy [33, 55].

2.1.1 Electric Circuit

Fig. 2.1 shows two electric circuits for modeling a synapse and a neural soma. First, in the dashed triangle, the current I can be $I_{syn} + I_{Inj}$, where I_{syn} is the synaptic current from other presynaptic neurons, I_{Inj} can be the summation of all extra current stimuli or light stimuli. In the circuit, I is separated into two currents I_R and I_C. I_R can be calculated be Ohm's law $I_R = V/R$. I_C can be calculated by $I_C = CdV/dt$. According to Kirhhoff's Current Law,

$$I(t) = C_m \frac{dV}{dt} + \frac{V(t)}{R} \tag{2.1}$$

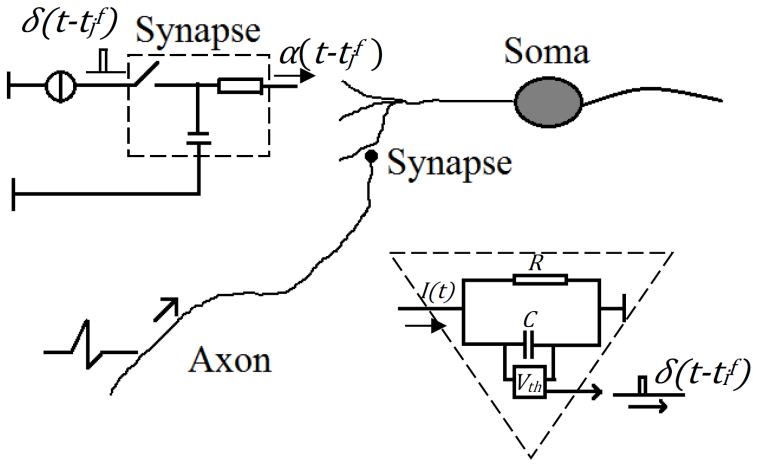


FIGURE 2.1: Integrate-and-Fire neuron model. The basic circuit of the synapse is the module inside the dashed square. A spike $\delta(t - t_j^f)$ is filtered and generates an input current alpha pulse $\alpha(t - t_j^f)$ at the synapse. The basic circuit of the postsynaptic neuron is the module inside the dashed triangle. The current $I(t)$ charges the RC circuit. When the voltage at the capacitance if over a threshold V_{th}. It generates an output pulse $\delta(t - t_i^f)$.

After we multiply both sides of Eqn. 2.1 by R and replace RC by constants τ_m, LIF equation can be written as,

$$\tau_m \frac{dV(t)}{dt} = -V(t) + RI(t) \tag{2.2}$$

where τ_m is the membrane time constant, V_m is the membrane potential, R is the resistance. Whenever V is larger than its threshold value V_{th}, the voltage is reset to its resting potential.

A more general form of LIF neuron is,

$$\tau_m \frac{dV}{dt} = (E_L - V) + F(V) + R_m I_{stim} \tag{2.3}$$

For linear model in Eqn. 2.2, $F(V) = 0$. For the quadratic integrate-and-fire model (QIAF)

$$F(V) = \frac{(V - E_L)^2}{\Delta_Q} \tag{2.4}$$

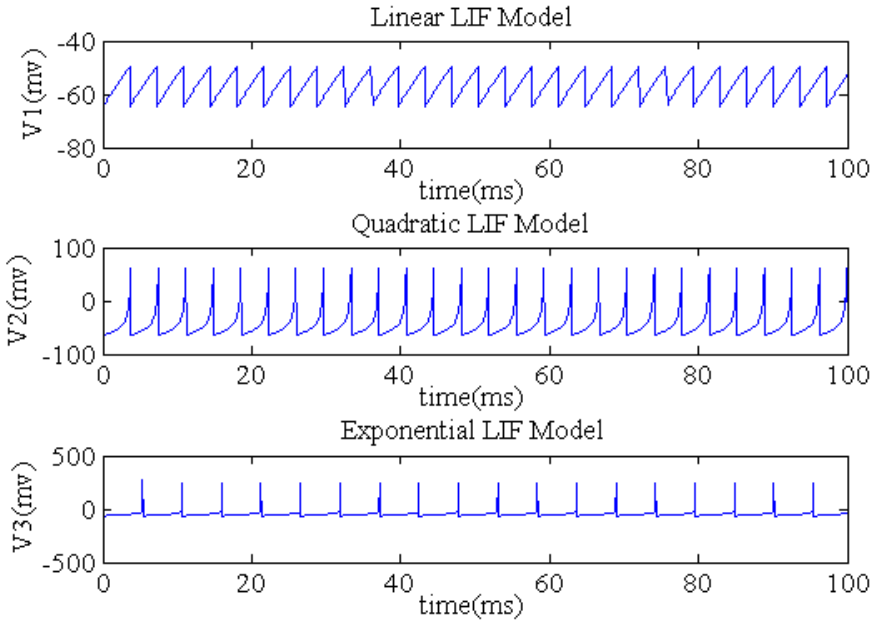


FIGURE 2.2: Membrane potential of linear, quadratic, and exponential integrate-and-fire neurons when stimulated by constant currents.

While for the exponential integrate-and-fire (EIAF),

$$F(V) = \Delta_E exp(\frac{(V - V_r)}{\Delta_E}) \tag{2.5}$$

where Δ_Q, Δ_E, V_T are constant voltages. Fig. 2.2 compares the membrane potential of three types of neurons stimulated by constant current. Fig. 2.3 shows the dV/dt relationship with V for three models.

2.1.2 Refractory Period

In the HH model, the action potential is mainly caused by the dynamics of sodium and potassium ions, which are governed by the opening and closing of their corresponding ion channels. In Fig. 2.4, during resting potential, both sodium and potassium channels are closed as in left bottom subfigure. When there is an excitatory synaptic current or an extra stimulus, the voltage gated ion channel Na^+ opens

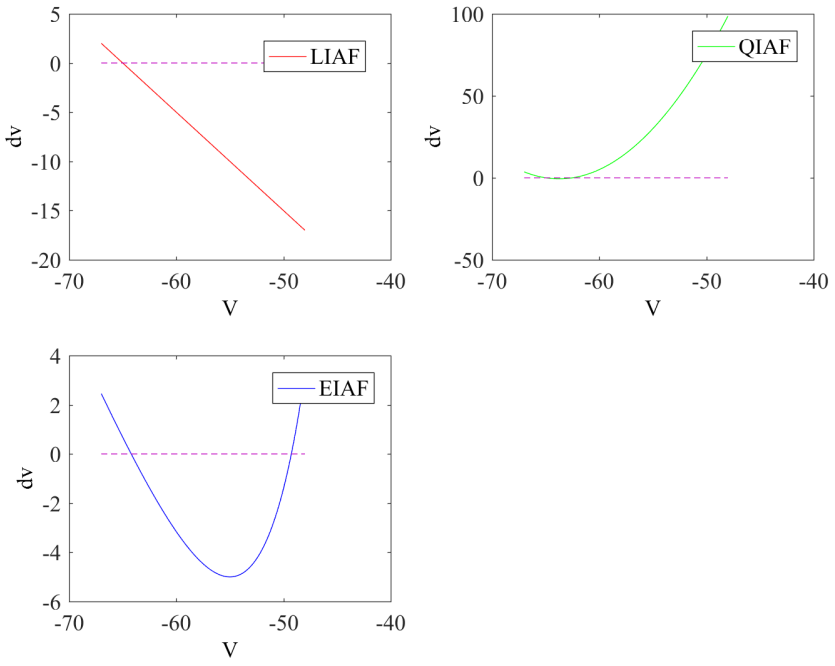


FIGURE 2.3: Dynamic analysis of different LIF models.

instantaneously as in left top subfigure. The sodium ions flux into the neuron, which cause the depolarization of the neuron. The sodium channels also close very fast after the membrane potential reaches the maximum in top right subfigure. Because the potassium channels open and close slower than the sodium channels, potassium ions flow out of the neuron even though the membrane potential decreases below the resting potential in bottom right subfigure.

There are two types of refractory periods: the relative refractory period and the absolute refractory period. The absolute refractory period is a period of time immediately after an action potential when it is impossible to have a second action potential no matter how large the second stimulus. The relative refractory period is a period when it is possible to generate an action potential by larger extra stimulus than the previous one. In Fig. 2.5, the second stimulus causes the increase of conductance of the sodium channel. In addition, the amplitude of second increase

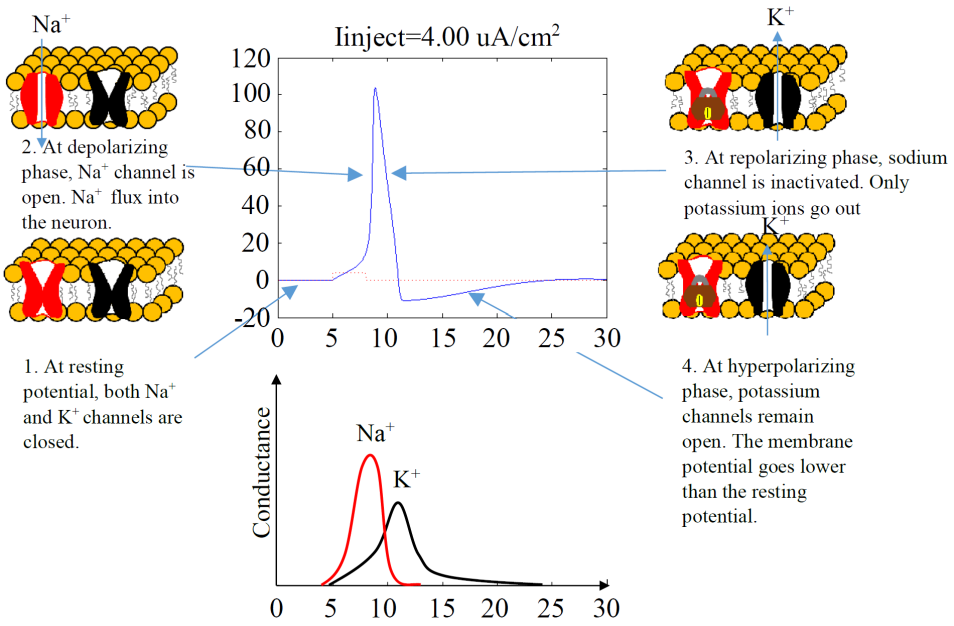


FIGURE 2.4: States of Na^+ and K^+ channels during action potential.

of the conductance is smaller than the first one even though its stimulus is larger. In Fig. 2.6, the second stimulus is very near the first stimulus, which therefore is the absolute refractory period. There is no change in the conductance of the sodium channel caused by the second stimulus.

2.2 Izhikevich Model

Even though LIF neurons are computational efficient, it is unrealistically simple and cannot produce rich spiking, bursting dynamics found in cortical neurons. Izhikevich proposed a new mathematical neuron model, which is both biologically plausible as Hodgkin-Huxley model and computational efficient as LIF neurons. By using four parameters, it can reproduce spiking and bursting behavior of most cortical neurons in Fig. 2.7.

By using Bifurcation methodologies [80], the four dimensional HH neuronal model can be reduced to two dimensional model in Eq. 2.6-2.7,

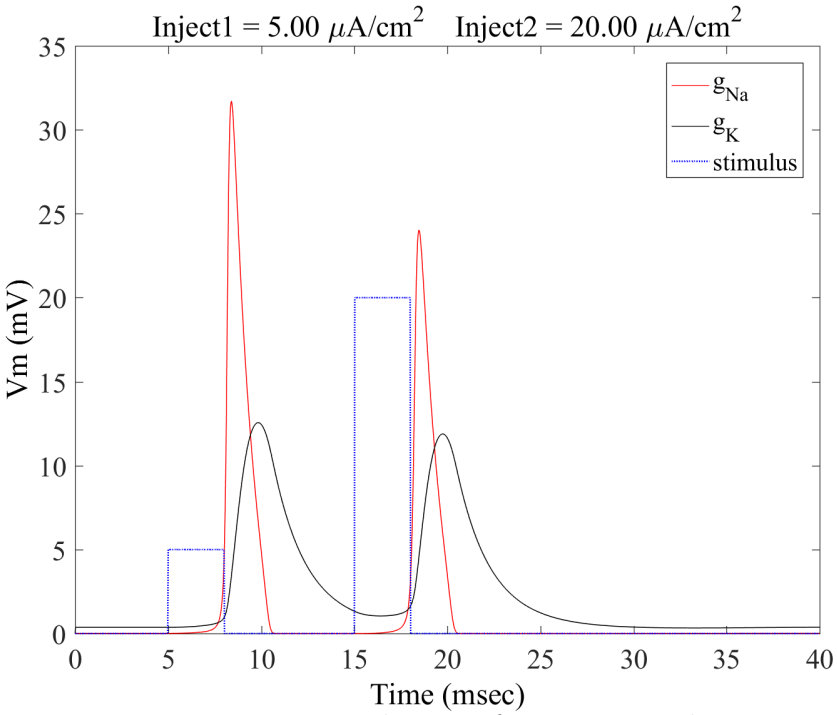


FIGURE 2.5: Relative refractory period.

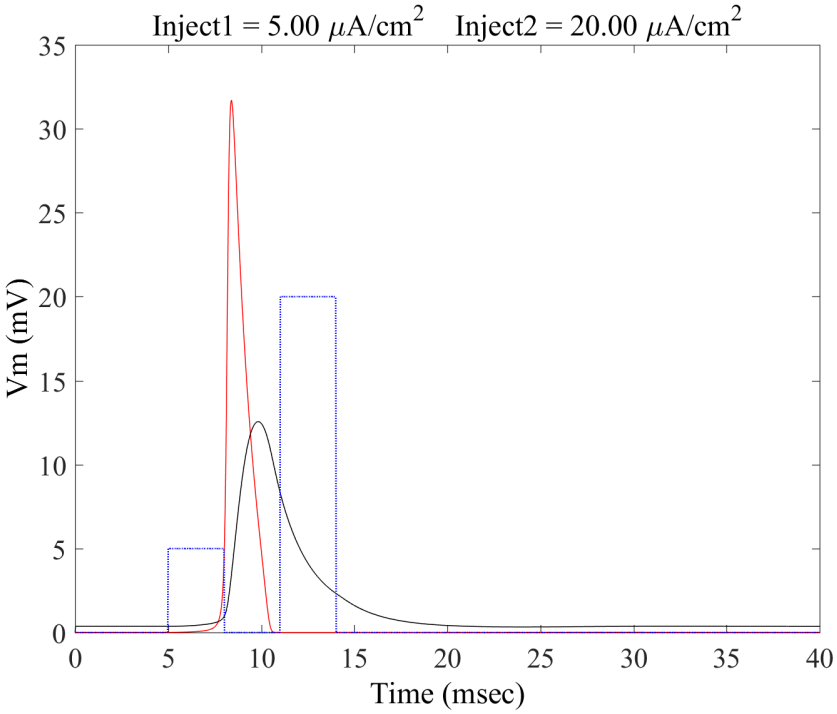


FIGURE 2.6: Absolute refractory period.

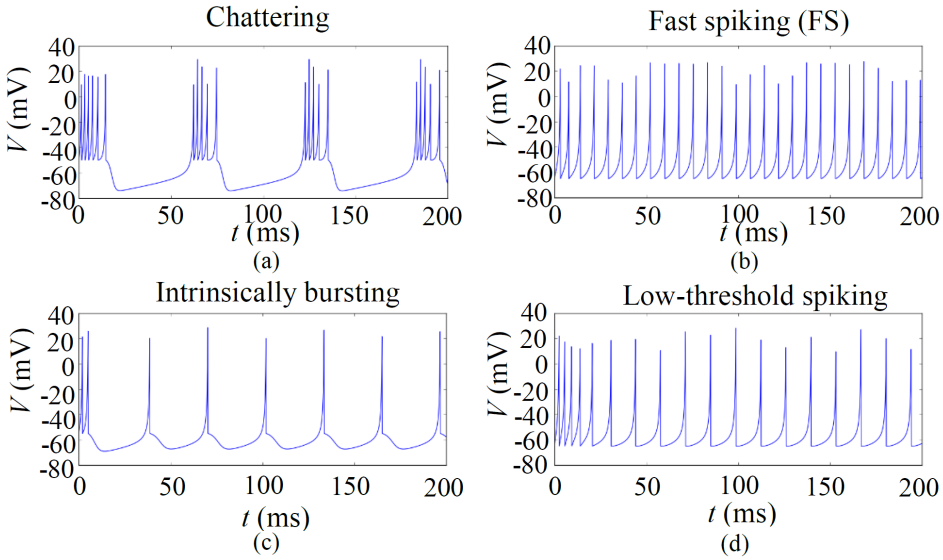


FIGURE 2.7: Different type of neurons with different values of a,b,c,d in the model described by the Eq. 2.6, Eq. 2.7. IB and CH are cortical excitatory neurons. FS and LTS are cortical inhibitory interneurons.

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

$$u' = a(bv - u) \tag{2.7}$$

with the auxiliary after-spike resetting

$$\text{if} \quad v \geqslant 30mV, \quad \text{then} \quad \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{2.8}$$

where v and u are dimensionless variables, a, b, c, d are dimensionless parameters, $' = d/dt$, t is the time. v is the membrane potential of the neuron while u is a membrane recovery variable, which can represent the activation of K^+ and inactivation of Na^+. So it's for decreasing v. If the spike reaches its maximum (+30mv), the membrane potential and recovery variable are reset based on Eq. 2.8. Synaptic currents and extra stimulus are represented by variable I. The coefficients in Eq. 2.6 are picked by fitting action potential of a cortical neuron, where v has the mV scale and t has

ms scale. a, b, c, d are defined as below:

- The parameter a determines the time scale of the recovery variable u. The typical value is $a = 0.02$.
- The parameter b determines the sensitivity of u to the subthreshold fluctuations of v. Higher values couple v, u more strongly leading to subthreshold oscillations and low-threshold spiking dynamics. It's typical value is 0.2.
- The parameter c determines the after-spike reset value of v. It's typical value is $c = -65mV$
- The parameter d determines the after-spike reset of u because of slow high-threshold Na^+ and K^+ conductances. It's typical value is 2.

Different parameters can result in many different types of neocortical [27, 67, 60] and thalamic neurons in Fig. 2.7. Fig. 2.7 shows several types of spiking and bursting in neocortical neurons of mammalian brains. Two excitatory cells include:

- In Fig. 2.7(a), CH (chattering) neurons fire stereotypical bursts. The inter-burst frequency can reach 40 Hz. For this model, $c = -50\ mV$ and $d = 2$.
- In Fig. 2.7(c), IB (intrinsically bursting) neurons fire a burst of spikes followed by repetitive single spikes. The parameters are $c = -55\ mV$ and $d = 4$. u builds up during initial burst and then switch to single spikes.

Two inhibitory cells are:

- In Fig. 2.7(b), FS (fast spiking) neurons can have periodic spikes with very high frequency without any adaption. In the model, $a = 0.1$.
- In Fig. 2.7(d), LTS (low-threshold spiking) neurons can also have periodic spikes with very high frequency but with frequency adaption. The neuron has low firing threshold by setting $b = 0.25$.

Izhikevich model can also simulate other types of excitatory neurons and inhibitory neurons such as RS (regular spiking) neurons, TC (thalamo-cortical) neurons, and RZ (resonator) neurons [79].

2.3 Synapse Models

When an action potential transmit to presynaptic terminal, the voltage dependent Ca^{2+} channels are opened. Ca^{2+} ions goes into the presynaptic terminal and bind on the vesicles that contain the *neurotransmitters*. Ca^{2+} causes vesicle migration, fusion to pre-synaptic membrane. Then the transmitters are released via exocytosis in quantized amounts. Transmitters diffuse across cleft toward post-synaptic membrane. Transmitters binds to receptor sites, which changes the membrane permeability of ions. This change can cause post-synaptic potential (PSP). It can be either excitatory (EPSP) or inhibitory (IPSP) depends on the synapse type. If EPSP exceeds threshold, action potential of postsynaptic neuron is generated. Transmitters are broken down by enzyme in cleft. The product are taken up by presynaptic site to generate new neurotransmitters.

Because many physiological processes happen during synaptic transmission, precise modeling of this transmission is challenging. In addition, the properties of synapses change on many different timescale making synaptic transmission is a highly dynamic process. Additionally, synaptic transmissions are believed to be one of the main source of noises in neural system. Even a single neuron can have thousands of synapses. Therefore, efficient simulation by using simplified computational models are essential.

Studies estimate that the average adult human brain has about 86 billion neurons [3]. Because a single neuron can have thousands of synapses, the estimated number of synapses is in trillions. Even if all synapses were identical, modeling the whole brain in real time can be an impossible task. Therefore, researchers developed dif-

ferent types of simplified synapse models including delta pulses, alpha synapses, and chemical synapses.

2.3.1 Delta Pulse

The delta pulse is the simplest model for modeling the synaptic current, which is defined as,

$$I_{syn} = \delta(t - t_f) \tag{2.9}$$

where $\delta(\cdot)$ is a Dirac delta function, t_f is the firing time of presynaptic neuron, I_{syn} is the synaptic current. It can be thought of as a function on real line which is zero everywhere except at the origin, where it is infinite,

$$\delta(t - t_f) = \begin{cases} +\infty, & t - t_f = 0 \\ 0, & t - t_f \neq 0 \end{cases} \tag{2.10}$$

This equation can satisfy

$$\int_{-\infty}^{\infty} \delta(t - t_f)dt = 1 \tag{2.11}$$

Synapses modeled by delta pulse can cause an instantaneous increase of the membrane potential of postsynaptic neuron like Fig. 2.8. To better illustrate this synapse model, the membrane potentials of two connected neurons are plotted in Fig. 2.8. Neuron 1 is the presynaptic neuron while neuron 2 is the postsynaptic neuron. Neuron 1 is stimulated by extra currents. The red dots in Fig. 2.8. (b) show the firings of the presynaptic neuron 1. In Fig. 2.8(c), we can see an instantaneous increase of membrane potential of neuron 2 when neuron 1 fires. The following decay after the instantaneous rise is due to the leaky integrate and fire model. The third spike of neuron 1 cause another instantaneous increase of V_2, which generates an action potential.

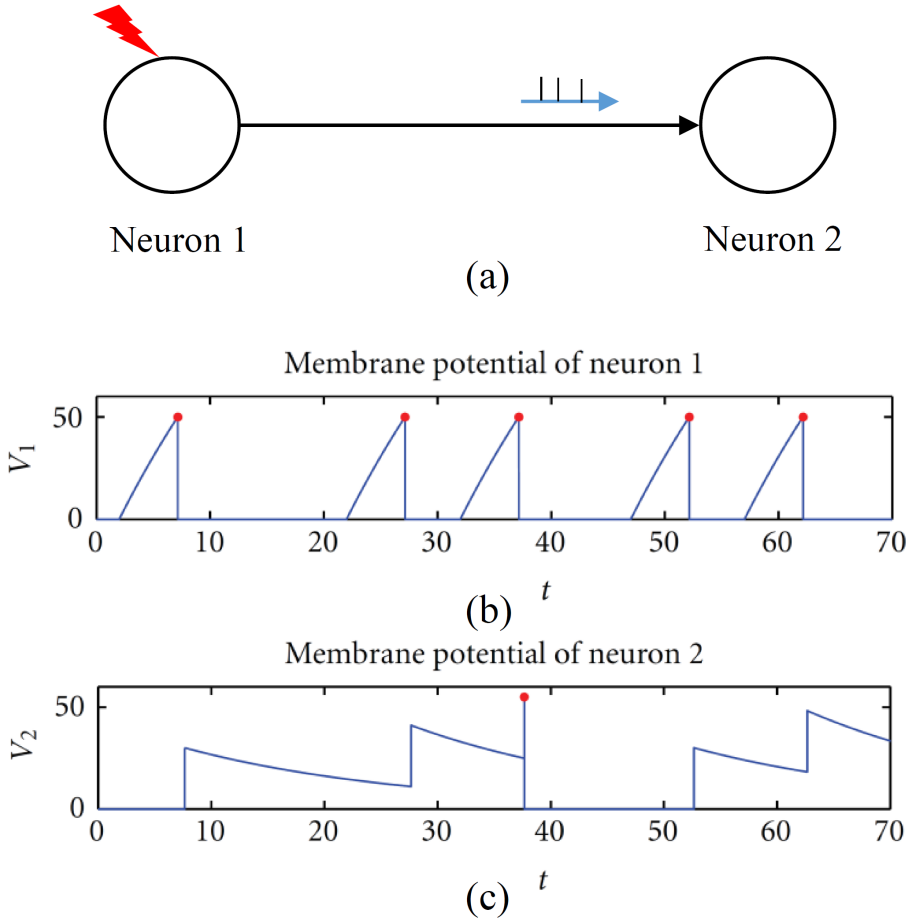


FIGURE 2.8: Membrane potentials of pre- and postsynaptic neuron by modeling synaptic current using delta pulse. (a). simple network structure. (b). membrane potential of presynaptic neuron stimulated by extra currents. (c). membrane potential of postsynaptic neuron.

2.3.2 Alpha Synapse

One weakness of delta pulse is that it does not capture the rising phase and decaying phase of synaptic conductances, which have strong effects on network dynamics [161]. Alpha function models both the rising phase and decaying phase of the conductance.

$$g_{syn}(t) = \bar{g}_{syn} f(e^{-(t-t_f)/\tau_{decay}} - e^{-(t-t_f)/\tau_{rise}}) \tag{2.12}$$

where g_{syn} is the synaptic conductance, t_f is the firing time, $\tau_{decay}, \tau_{rise}$ are time constants for the decaying speed and rising speed of conductance respectively, f is

the normalization factor to ensure that the amplitude is $\bar{g}_{syn}$,

$$t_{peak} = t_0 + \frac{\tau_{decay}\tau_{rise}}{\tau_{decay} - \tau_{rise}} ln(\frac{\tau_{decay}}{\tau_{rise}}) \tag{2.13}$$

The normalization factor f is defined by,

$$f = \frac{1}{-e^{-(t_{peak}-t_f)/\tau_{rise}} + e^{-(t_{peak}-t_0)/\tau_{decay}}}. \tag{2.14}$$

The weakness of using the equations above to model the conductance change is that all the inputs must be tracked and stored. Another method that does not need storing spike times is by solving the two coupled linear differential equations for the synaptic conductance [168],

$$g_{syn}(t) = \bar{g}_{syn} f g(t) \tag{2.15}$$

$$\frac{dg}{dt} = -\frac{g}{\tau_{decay}} + h \tag{2.16}$$

$$\frac{dh}{dt} = -\frac{h}{\tau_{rise}} + h_0 \delta(t_f - t) \tag{2.17}$$

where h_0 is a scaling constant. In Fig. 2.9-Fig. 2.10, most time courses of synaptic conductances can be well modeled by these two equations. In Fig. 2.10, the left subfigure shows the coupled ODE simulated GABAa synaptic conductance comparing with summation of alpha functions while the right subfigures presents the comparison of coupled ODE simulated NMDA and summation of alpha function for NMDA. In general, the coupled ODE models the alpha function very good and efficient.

2.4 Spike Timing-Dependent Plasticity (STDP)

In most neuron models, each synapse is described by a constant parameter w_{ij}, which defines the amplitude of postsynaptic response to an synaptic current. According to some electrophysiological experiments, these amplitudes are not fixed but can change

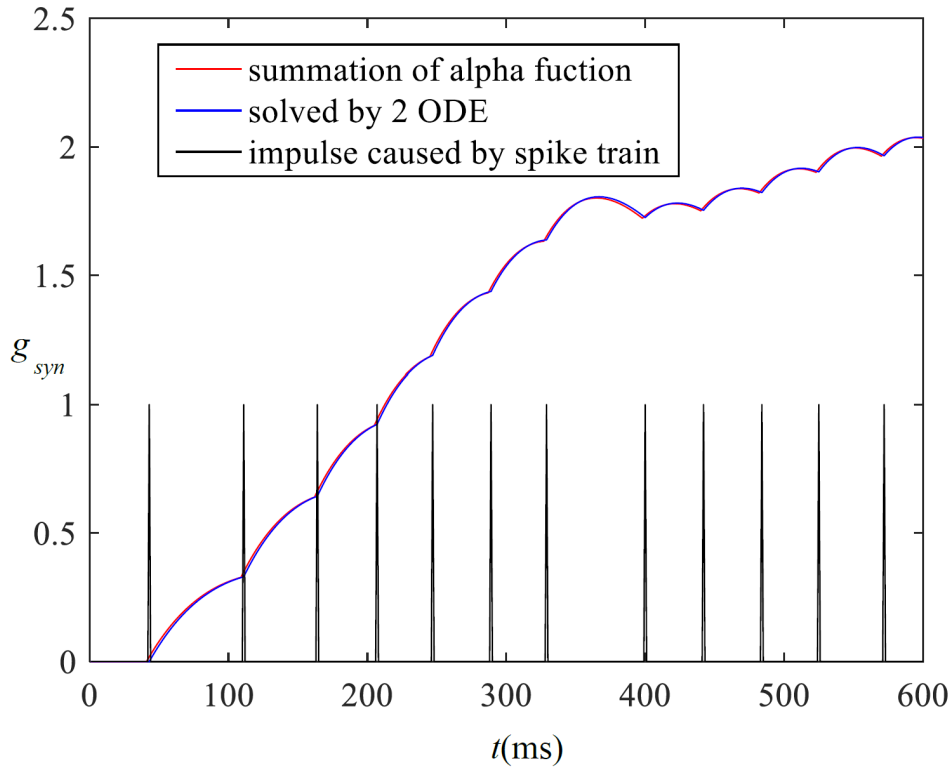


FIGURE 2.9: Conductance simulated by summation of alpha functions and coupled ODEs.

over time. Proper electrical or optic stimulus can make changes of the synaptic weights for hours or days. These experiments are mostly inspired by Hebb's postulate ([70]):

> When an axon of cell A is near enough to excite cell B or repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

Experiments for proving Hebb's postulation found long-lasting changes in synaptic weights [12]. If stimulations cause persistent increase of synaptic weights, the effect can be called long-term potentiation (LTP). If the synaptic weight is decreased, it can be called long-term depression (LTD). However, Hebb didn't define clearly

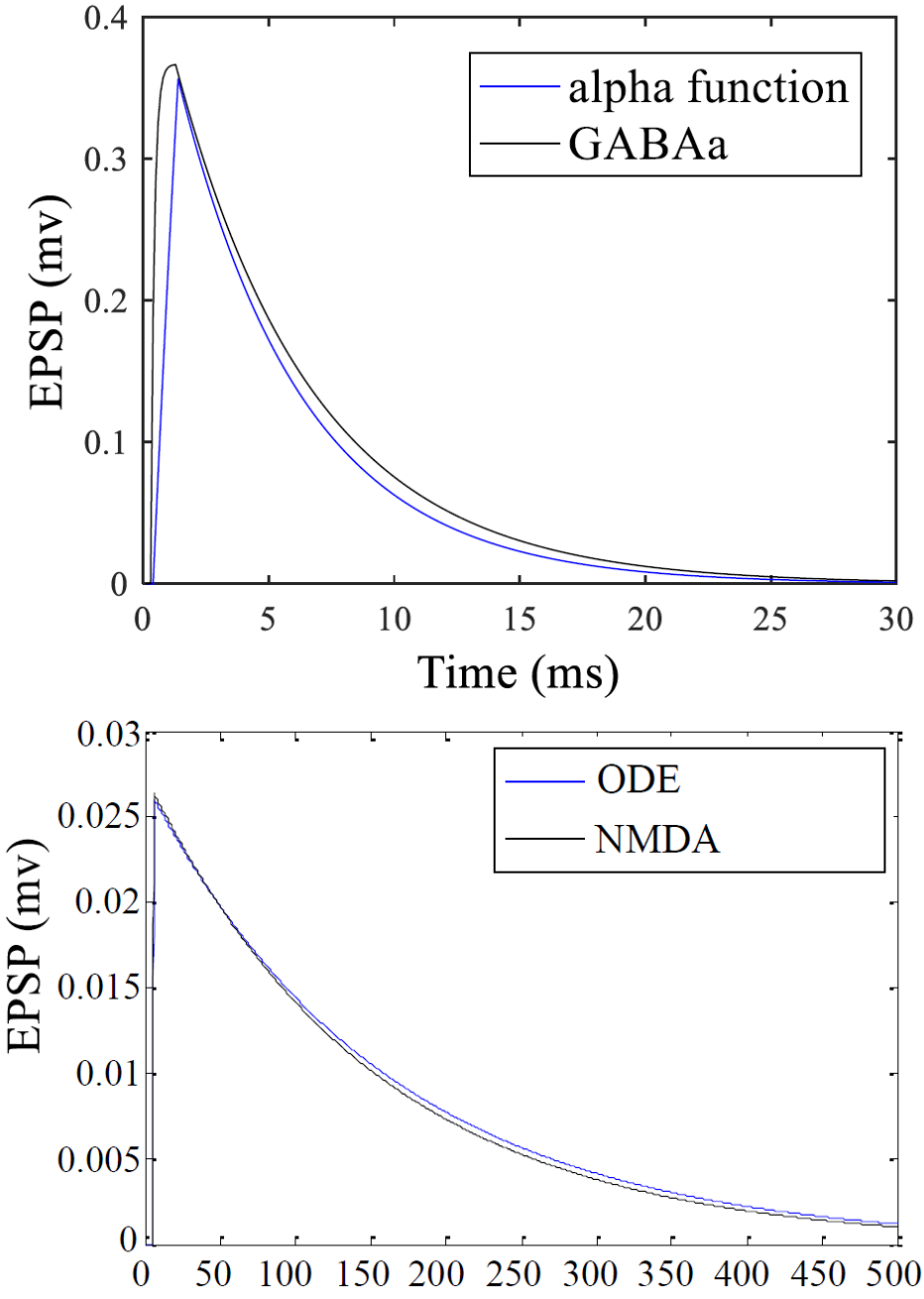


FIGURE 2.10: Conductance simulated by alpha function and coupled ODE for GABAa and NMDA respectively.

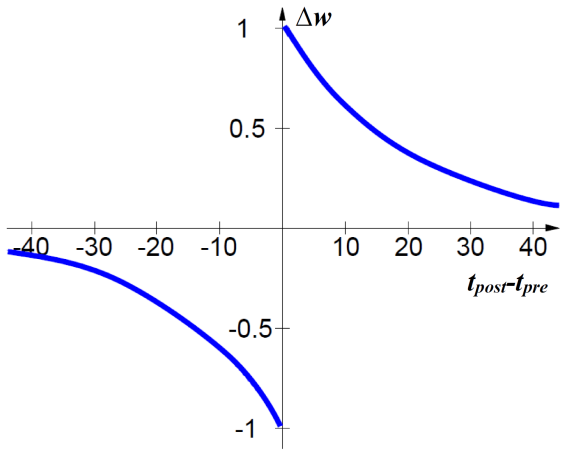


FIGURE 2.11: STDP learning scheme.

how the firing of pre- and postsynaptic neuron can be correlated. Spike Timing Dependent Plasticity (STDP) is found to be a temporally asymmetric form of Hebbian learning with precise definition of the temporal correlations between the spikes of pre- and postsynaptic neurons [10]. The basic STDP model is usually defined by,

$$W(\Delta t) = \begin{cases} A_+ exp(-(t_{post} - t_{pre})/\tau_+), & if \quad t_{pre} < t_{post} \\ -A_- exp((t_{post} - t_{pre})/\tau_-), & if \quad t_{pre} > t_{post} \end{cases} \tag{2.18}$$

where Δt means the temporal difference between the presynaptic spike and postsynaptic spike, A_+, A_- are the maximum increase and decrease, τ_+, τ_- are two time constants. Fig. 2.11 shows the weight change defined by STDP rule. If there are multiple spikes from both presynaptic neuron and postsynaptic neuron, many different paring schemes exist to pair those spikes [119].

2.4.1 Implement STDP using Local Variables

STDP update rule can be implemented using two local variables so that the memory doesn't need to track all spike timings. One variable is for a low-pass filtered version of the presynaptic spike train. The other is for the postsynaptic spike train. Each spike of presynaptic neuron i contribute to the value of x_i:

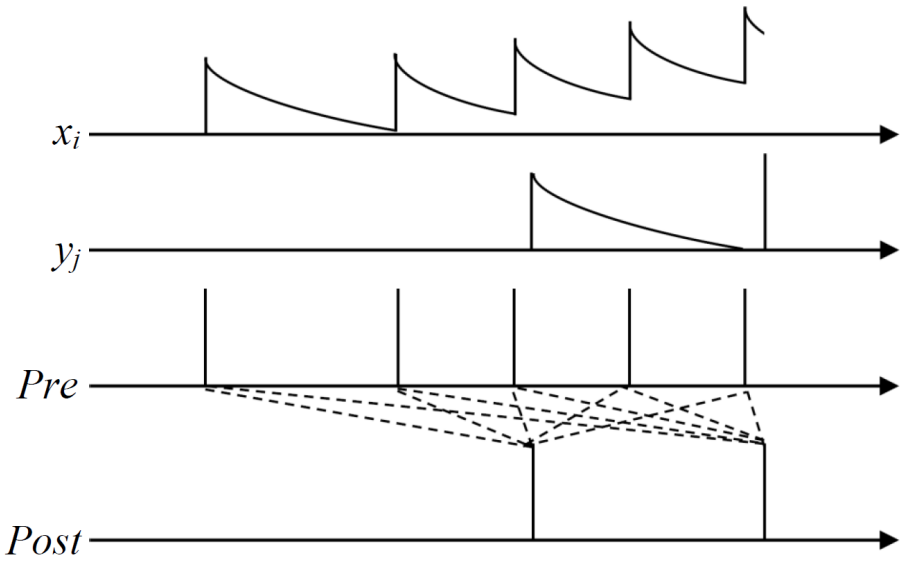


FIGURE 2.12: Local variables x_i, y_i and their pairings.

$$\frac{dx_i}{dt} = -\frac{x_i}{\tau_x} + \sum_{t_i^f} \delta(t - t_i^f) \tag{2.19}$$

where t_i^f is the firing times of the presynaptic neuron. x_i increase by 1 each time when i spikes and decrease with time constant τ_x. For the postsynaptic neuron j, its firings contributes to variable y_j:

$$\frac{dy_j}{dt} = -\frac{y_j}{\tau_y} + \sum_{t_j^f} \delta(t - t_j^f) \tag{2.20}$$

where t_j is the spike timing of postsynaptic neuron j. The dynamics of the local variable x_i and y_j can be seen in Fig. 2.12. The weight change is governed by,

$$\frac{dw_{ij}}{dt} = -A_- y_j(t)\delta(t - t_i^f) + A_+ x_i(t)\delta(t - t_j^f). \tag{2.21}$$

Eq. 2.21 describes the all-to-all pairing. Another pairing scheme is called nearest-neighbor in Fig. 2.13. In Nearest-Neighbor pairing, each presynaptic neuron's spike

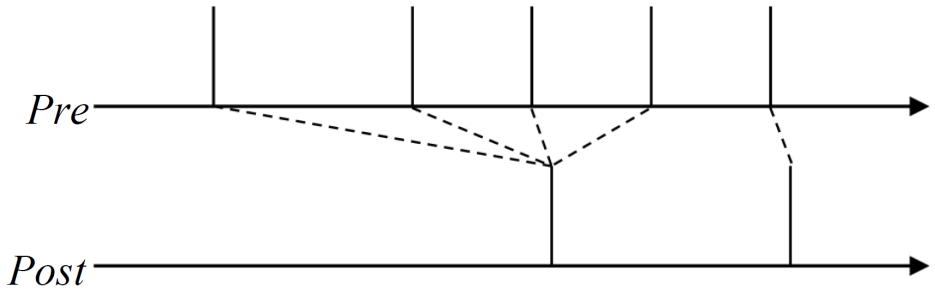


FIGURE 2.13: Nearest-Neighbor pairing scheme

only pairs with its nearest spike of the postsynaptic neuron. It can be realized using proper local variables. The local variables are reset to 1 other than increase by 1 in all-to-all pairing scheme.

2.5 Discussion

In this thesis, two main types of neural models are used including Integrate-and-Fire model and Izhikevich model. Integrate-and-Fire model, as a one dimensional model, is one of the most efficient neuron models used for solving classification, pattern recognition and control problems, which require real time processing and fast feedback signals such as hand writing digit recognition, autonomous driving and spam classification. In my work, LIF neurons are used for training SNNs to control unicycle modeled virtual insects using indirect perturbation algorithm and indirect SGD algorithm. They are simulated using CSIM simulator, which can simulate SNNs with complex 3D structures and connection distributions. The trained insects can navigate in an unknown environment and find its target while avoiding obstacles.

In order to satisfy our algorithms motivation of training biological neuron networks, the neural network's model have to be more biologically plausible so that what happens in the biological neuronal system and culture can be simulated. Therefore, as a two dimensional, Izhikevich model is used in my thesis for training SNNs to control both robot in real world environments. As a GPU accelerated SNN simula-

tor, CarlSim is used for simulating large size SNNs with Izhikevich neurons in real times. The main advantage of CarlSim comparing with other simulators is that it can simulate large size neural networks based on CUDA on GPU, which enable us to run the simulation of SNNs in real time. Comparing with LIF neuron, Izhikevich neuron model can model different types of firing patterns, which usually can be found in real neural systems and neuronal cultures.

In this thesis, alpha synapse model with both rising phase and decaying phase is used to model the shape of conductance change during the synaptic transmission. By using different parameters of the decaying and rising constants, different types of biological synapses can be simulated in a good precision such as NMDA, GABAa and GABAb. The magnitude of the synaptic conductance is a function of the time difference between the spiking of the pre-synaptic neuron and the spiking of the post-synaptic neuron, a model of spike-time-dependent-plasticity (STDP). While the nature of synaptic plasticity in the cockroach has yet to be determined, STDP has been observed in many biological preparations. Comparing with a lot of rate-based hebbian learning rule, the STDP depends on precise timings of the pre and postsynaptic activities, which has been found in different types of biological neuronal systems. In order to modeling it in a fast speed, this thesis implement it using local variables so that the memory do not need to save every spikes in the past by only save the value of those two local variables, which changes whenever the pre or postsynaptic neuron fires.

3

Problem Formulations

There have been a number of studies that have shown that a feed-forward network with one hidden layer consisting of a finite number of neurons can approximate continuous functions on compact subsets of $\Re^n$ after training [30, 71]. This is often referred to as the *universal approximation theorem*. Recurrent neural networks are even more powerful because it has been proven that they can approximate any dynamical system [54]. Spiking neural network, the third generation of neural networks, has been proven to solve the same problem as ANNs using fewer neurons [100]. An SNN can be classified according to its coding schemes. Rate coding and temporal coding are two coding schemes that SNNs can use.

3.1 Temporal and Spatial Mapping using Temporal coding

One of a SNN's properties is the ability to both encode and decode information in temporal form, which is hard to do in ANNs. Fig. 3.1 shows the comparison of coding of information in ANNs and SNNs. In Fig. 3.1(a), the information flows in ANNs all contain real values with various amplitudes. The information given to the ANNs are encoded by the input neurons into analog signals at the same time when

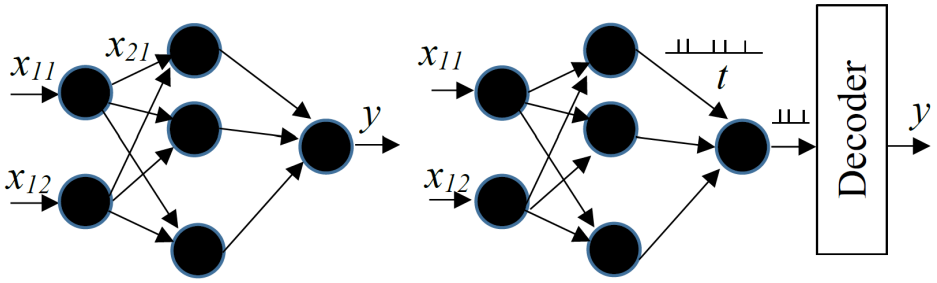


FIGURE 3.1: Information flow of ANNs comparing with SNNs.

the input is given. In contrast, in Fig. 3.1(b), information given to the SNNs is encoded by the input neurons into spike trains with different timings. All the action potentials have almost the same shape. Therefore, the timings or the rate of the spikes actually matters because they are the way that SNNs use to store, compute and transmit information.

A long-standing debate within neuroscience community is whether the nervous system use temporal coding or rate coding. In motor neurons, the strength at which muscles is flexed depends only on the firing rate. However, external stimulus in visual and auditory systems are found to be coded by temporal coding schemes [22, 56]. Most existing training algorithms are designed for a specific coding scheme [51, 14]. The indirect training algorithms developed in this thesis are tested on both temporal coding and rate coding problems. A commonly test case for temporal coding is the classification of input spike trains into different classes according to the outputs. The basic structure of the network can be seen in Fig. 3.2 [156].

m input synapses are used for each output neuron. An input layer with m input neurons is all-to-all connected to the input synapses of each output neuron.

3.1.1 Correlation-Based Metric

As proposed in [140], a correlation-based metric (C) measures the similarity between two spike trains. Similarity coefficient C is one for two same spike trains and zero

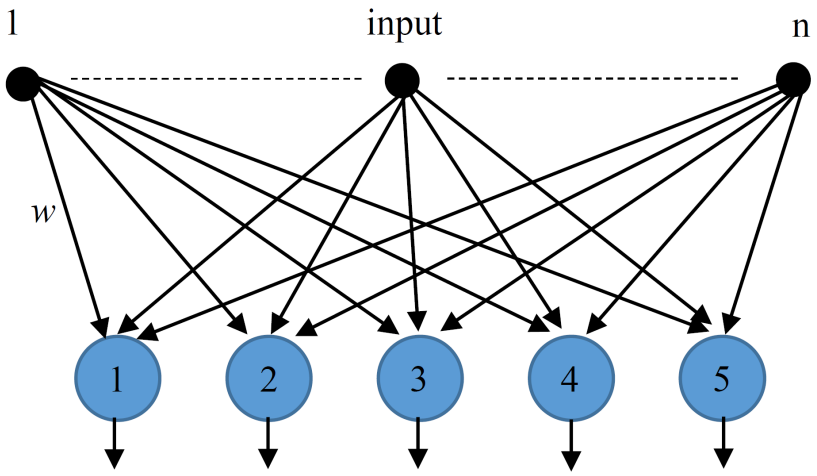


FIGURE 3.2: Each neuron is trained to classify one type of spike train.

for two uncorrelated spike trains. C is defined as,

$$C = \frac{v_d \cdot v_0}{|v_d||v_0|} \tag{3.1}$$

where v_d, v_0 are two vectors of convolutions created from a desired spike train and actual output spike train. The convolution transforms series of Dirac delta functions into continuous function. Symmetric Gaussian function is used to convolve with the spike trains in Eq. 3.2

$$f(t) = e^{\frac{-t^2}{e^{2\delta^2}}} \tag{3.2}$$

where δ is a constant that governs the width of the symmetric Gaussian function.

3.1.2 Training and testing Data

Every spatiotemporal input pattern has m spike trains with time length of T ms. Nt samples are generated in each of the five different classes by adding a Gaussian jitter with a standard deviation to the base pattern. Gaussian jitter is a type of data signal jitter that is unpredictable electronic timing noise. It follows Gaussian distribution because most noise or jitter in an electrical circuit is caused by thermal

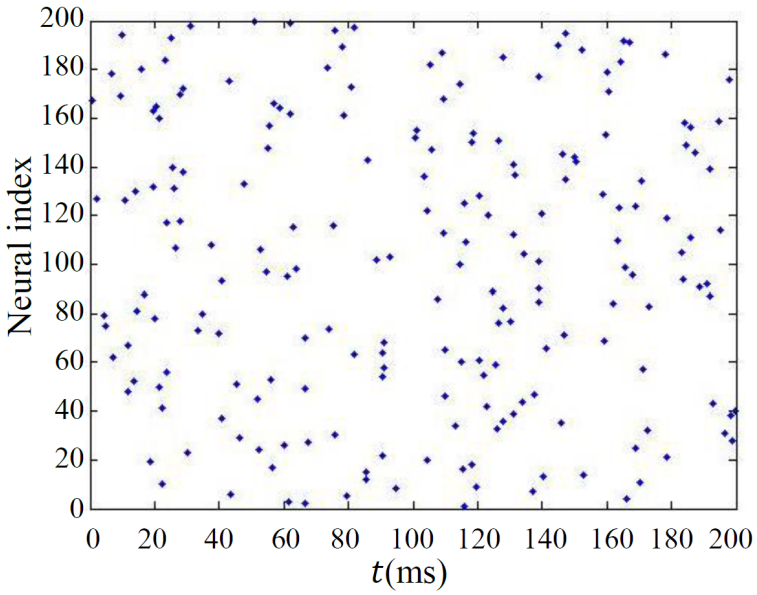


FIGURE 3.3: One input data sample generated according to Eq. 3.3

noise with gaussian distribution. The time of input spike after adding jitter can be written as,

$$T_i = UniformRand_i + GaussianRand_i \tag{3.3}$$

T_i is the input spike timing for neuron i, $UniformRand_i$ is the random time picked from uniform distribution, $GaussianRand$ is the random number picked from Gaussian jitter. Fig. 3.3 shows an example of input generated by Eq. 3.3.

Fig. 3.3 is randomly generated according to one base pattern and one gaussian jitter. The base pattern is the randomly generated term in Eq. 3.3 [123]. Another set of data for testing is generated in the same way. The final data set for training is classified to five classes according to five different base pattern. Each class has 15 data samples generated by adding the Gaussian jitter.

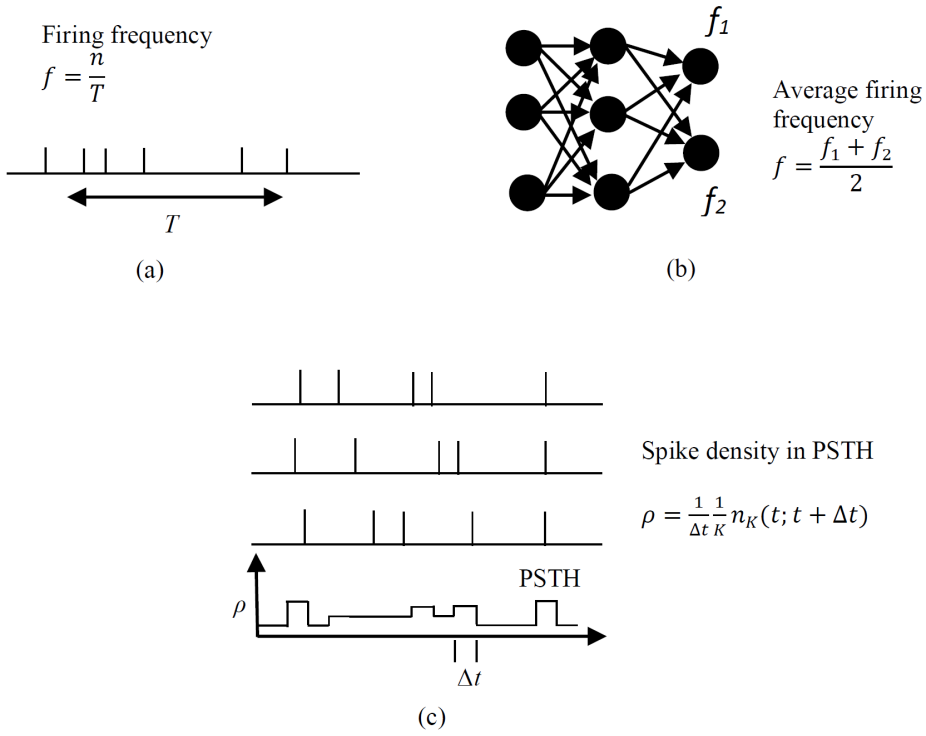


FIGURE 3.4: Different rate coding methods. (a) rate coding over time of single neuron. (b) rate coding over a population of neurons. (c) rate coding over different trials.

3.2 Input Output Mapping using Rate Coding

Rate coding assumes that most information from the stimulus is encoded into the firing rate of the neurons. Input noise can cause big variation of specific timings of the spikes. Therefore, rate coding is very robust to inter-spike interval (ISI) noise [151]. Precise calculation of firing rate is essential during simulation. There are several different definitions of firing rates according to different average procedures. For example, in Fig. 3.4(a), average firing rate over time is different from average of firing rate over different trials in Fig. 3.4(c). The average over a population of neurons is shown in Fig. 3.4(b).

The input to the neural network can either be spike trains or a current stimulus. The output of the SNN is decoded using the rate coding strategy described previously.

The training data set is given to the input layer of the SNN while the output layer is used to decode the spikes into firing frequencies.

This input and output mapping task can be described by,

$$y(t_k) = g_l[\xi(t_k)] \leftarrow SNN[\xi(t_k), W_l(t_k)] \triangleq \hat{y}(t_k) \tag{3.4}$$

where y is the desired function value, $\hat{y}(t_k)$ is the decoded output from an SNN, $\xi(t_k)$ is the input at time interval $[t_k, t_{k+1}]$, g_l is the function that the SNN needs to map.

3.3 Neural Network Size and Inputs Noise

Neural networks have been used for noise reduction both in speech recognition problem and image classification problems [141, 97, 180]. One of the strengths of convolutional neural networks compared with other machine learning methods for image recognition are their ability to reduce noise. Support Vector Machines accomplish this be preprocessing the images by de-skewing [26]. The size of the network needed to solve a given task is expected to change depending on the amount of noise. In this thesis I explored the effect of network size for different input noises for robots with the same initial position and target positions.

3.4 Discussion

This chapter describes the types of problems solved by the four indirect training algorithms used on SNNs so that the reader can generalize which indirect algorithms to use when new problems are encountered. The problems are classified as temporal and spatial mapping problems via temporal coding and input output mapping via rate coding. The temporal and spatial mapping problems can only be solved via spiking neural networks because of its dependence on complex information transmission

modeling and coding requirements. Algorithms that are applied to train rate based neural networks like back-propagation cannot be directly applied to solve spatial and temporal mapping problems. Therefore, specific algorithms have been developed for solving this type of problems such as ReSuMe, DL-ReSuMe and tempotron. However, the learning rules to update weights are not biologically plausible enough to be implemented in real neuronal networks. Therefore, the indirect ReSuMe algorithm described in this thesis is developed to solve this type of problems by adapting the ReSuMe algorithm to implement the STDP learning rule and indirect stimulation via square pulses. The new algorithm has potential ability to be applied in real neural systems.

Input output mapping by rate coding problems include training of virtual insects and neurorobots for navigating in unknown terrains to find targets while avoiding obstacles. Rate coding is not only used in Spiking Neural Networks but also in ANNs, in which the state of the neuron can be interpreted as the firing rate of the neuron. In addition, some parts of the human or animal's biological neuronal systems are based on rate coding like muscle force. In addition, the image projected onto the retinal photoreceptors changes therefore every few hundred milliseconds. For the virtual insect navigation problem and neurorobot with *Optitrack*, the inputs given to the network are the sensory information according to the distance from the insect's left and right target sensors to the target. The outputs are the firing rates of the output neurons in the corresponding networks, which drive the motors of the insects and robots. For a neurorobot with an embedded camera, the input given to the network is the image captured by the camera while the outputs are the firing frequencies of the output neurons, which control the moving speeds of the robot wheels. The mapping of sensory information to the desired output is created by indirect training using rate codings.

Further analysis of the algorithms is performed by comparing virtual insect's

performances controlled by different sizes of networks trained by indirect algorithms. This can help in determining the size of the network needed for a given problems. In addition, sensory noise cannot be neglected if the robot is used to solve problems in real world environment. Therefore, the behaviors of insects and robots under different sensory noise need to be analyzed so that those algorithms can be implemented robustly.

4

Indirect Training Algorithms based on Rate Coding and Applications

Indirect Training involves using stimulation to force the neural network to fire in a pattern that can lead to a change in the synaptic weights according to its own biologically plausible learning rule. Most spiking neural network training paradigms such as Spikeprop, genetic algorithm and self-organization directly set the synaptic weights based on their weight update formulas. These direct methods have been shown to train networks to solve both classification and regression problems. However, they are not biologically implementable in vivo or in vitro because the synaptic efficacy cannot be directly altered. Indirect training algorithms can be applied to neuromorphic chips, prosthetics and neuronal cultures.

In this thesis, I have developed four indirect methods to drive the synaptic weights to its desired value in a network through STDP, namely, Indirect Perturbation, Indirect Stochastic Gradient, Indirect ReSuMe, and Indirect Training with Supervised Teaching Signals. In the first three methods, the firing times of presynaptic neurons are tuned to satisfy the below constraints,

$$t_{i,k} \leqslant c_k + \frac{\beta}{2} \tag{4.1}$$

$$t_{i,k} + \Delta^{abs} \geqslant c_k + \frac{\beta}{2} \tag{4.2}$$

where $t_{i,k}$ is the k_{th} firing of neuron i, c_k is the k_{th} square pulse given to the neuron, β is the width of the square pulse, Δ^{abs} is the time length of the refractory period of neurons. Because all square pulse's heights and widths are the same, the firing time of presynaptic neuron given square pulses can be written as,

$$t_{i,k} = c_k - \frac{\beta}{2} + T \tag{4.3}$$

where $t_{i,k}$ is kth firing time of neuron i, T is the time costs for a neuron to fire given constant input height ω,

$$T = -\tau_m \ln\left[1 - \frac{\xi}{\omega R_m}\right] \quad (IR_m > \xi) \tag{4.4}$$

where $\xi = V_{th} - V_0$ is the voltage difference between the threshold and resting potential, R_m is the resistance, τ_m is the passive-membrane time constant. Therefore, T can be tuned by adjusting ω.

4.1 Indirect Perturbation Algorithm

4.1.1 Algorithm Description

We have previously shown that an SNN can be trained using, Indirect Perturbation, to control the virtual insect to accomplish the objectives of the navigation problem [177]. The training is achieved through the application of training stimuli given to the network at precise times which, in turn, indirectly modifies the synaptic weights via the internal STDP mechanism. The training signals are determined from the

SNN's response to a set of training data, which takes the form of $m = 6$ cases of sensor stimuli illustrated in Fig. 4.1 and the desired decoded output signals from the output neuron layer. Then the goal of the training becomes to reduce the error between the SNN's decoded output and the desired output provided by Eq. (4.5).

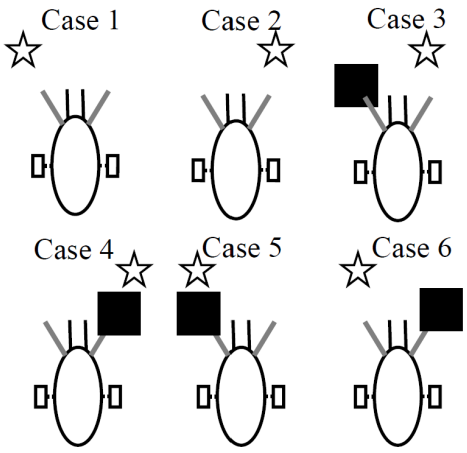


FIGURE 4.1: Insect locations for six training cases. The star represents the target while the black square represents the obstacle.

Based on the insect's objectives, the desired output signal of the SNN can be defined as,

$$\begin{pmatrix} f_L^* \\ f_R^* \end{pmatrix} = \begin{pmatrix} h_L & g_L \\ h_R & g_R \end{pmatrix} \begin{pmatrix} \kappa \\ \eta \end{pmatrix} \tag{4.5}$$

where f_L^* and f_R^* are the desired average firing frequencies of the left and right motor neurons, and h_L, g_L are the sensor input stimuli from Eq. (4.6). The constants κ and η are chosen such that $\kappa > \eta$ to prioritize obstacle avoidance over reaching the target.

$$\begin{aligned} g_L &= \alpha(d_L(x_L, y_L) + \lambda[d_L(x_L, y_L) - d_R(x_R, y_R)]) \\ g_R &= \alpha(d_R(x_R, y_R) + \lambda[d_R(x_R, y_R) - d_L(x_L, y_L)]) \end{aligned} \tag{4.6}$$

where the R and L subscripts of x and y correspond to the xy-positions of the right and left sensors, respectively, d_L and d_R are the Euclidean distances between the target position and the left and right target sensors in Fig. 4.2(b), respectively, $\lambda = 5$ is a constant value chosen heuristically to amplify the target sensor inputs differences between the left target sensor value and the right target sensor value, and $\alpha = 10^{-9}$ is the scaling factor of the sensor inputs.

h_R, and g_R are defined by Eq. (4.7). In Fig. 4.2(b), the terrain sensors' stimuli, defined as h_L and h_R for the left and right antennae, respectively, are governed by,

$$\begin{aligned} h_L &= \alpha\gamma\tfrac{\sigma}{r(x_L,y_L)+1} \\ h_R &= \alpha\gamma\tfrac{\sigma}{r(x_R,y_R)+1} \end{aligned} \tag{4.7}$$

where $\gamma = 0.1$ is a constant scalar chosen heuristically to bound the terrain sensor inputs, $r(x_L, y_L)$ is roughness in other words the pixel value at position (x_L, y_L) of the terrain image and $\sigma = 255$ denotes the maximum pixel value such that when $r = \sigma$, the measured terrain is flat.

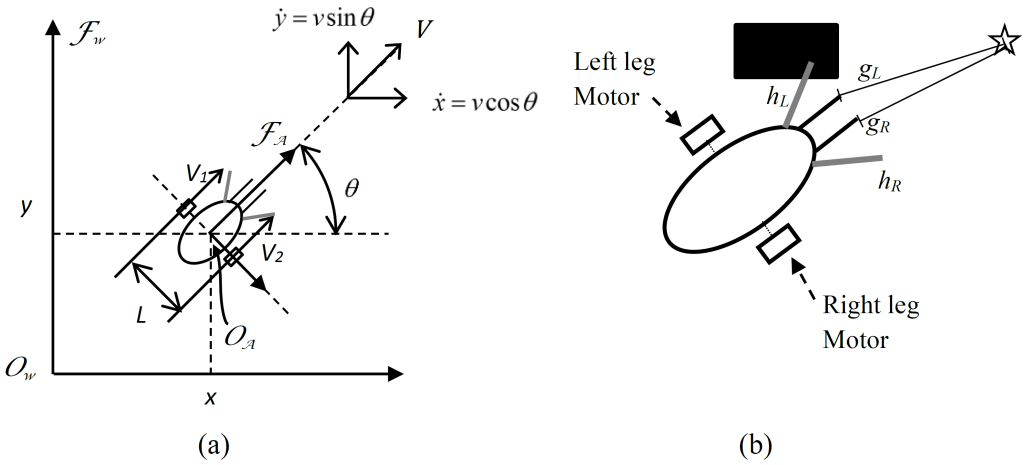


FIGURE 4.2: Virtual insect model. (a). Insect geometry and workspace coordinates. (b). Insect terrain sensors (gray) and target sensors (black).

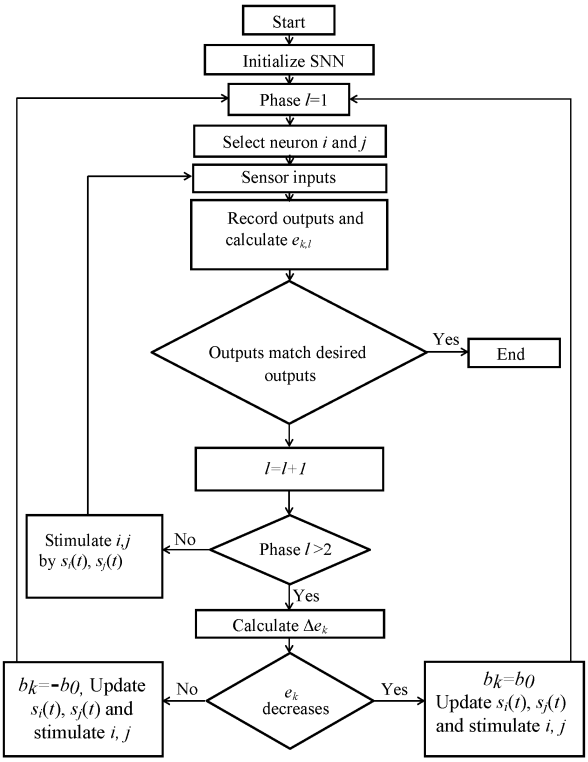


FIGURE 4.3: Flowchart of the training algorithm.

Each training iteration is divided into $n = \binom{N}{2}$ epochs, which is the number of possible ordered pairs of training neurons, where N is the number of input neurons. Batch training is performed on the SNN with only two input neurons receiving training stimuli per epoch. In addition, all neurons in the input layer receive their corresponding sensor stimuli during testing phases of each epoch. For each epoch, a pair of input neurons is selected from a list of n possible ordered pairs, denoted by $\mathcal{N}$, to receive training stimuli. No training signals are given to the other input neurons during this epoch.

For each of the $m = 6$ training cases, the input neurons are given signals that simulate the corresponding stimuli, $d_L(x_L, y_L)$, $d_R(x_R, y_R)$, $r(x_L, y_L)$, and $r(x_R, y_R)$, from the four insect sensor models Eq. (4.6) and Eq. (4.7). The inputs of the

training cases are a set of m vectors, denoted by $\mathbf{I}_i \subset \mathbb{R}^4, i = 1, ..., m$, each of which contains an instance of environmental information for the four insect sensors. Then the ℓ^{th} training error $e_{k,\ell},\ \ell = \{1, 2\}$, of the k^{th} epoch, can be evaluated from,

$$e_{k,\ell} = \frac{\sqrt{\sum_{i=1,...,m}[\mathbf{u}_i^* - \mathbf{u}_i]^T[\mathbf{u}_i^* - \mathbf{u}_i]}}{m} \tag{4.8}$$

where $\mathbf{u}_i = [f_R\ f_L]^T$ is the decoded motor control output of the SNN for training case i, $\mathbf{u}^* = [f_R^*\ f_L^*]^T$ is the desired output from Eq. (4.5), and ℓ is an index corresponding to the two testing phases per epoch.

Fig. 4.3 describes the process in one epoch. Each epoch consists of four phases in the following order: an initial testing phase, an initial training phase, a final testing phase, and a final training phase. The purpose of the first three phases is to evaluate the change in error, $\Delta e_k = e_{k,2} - e_{k,1}$, brought about by a test training signal (i.e., a series of stimuli) in the initial training phase. Then, during the final training phase, an extended training signal, which is a function of Δe_k, is given to the selected pair of input neurons to induce favorable training on the SNN.

Expanding on each of the four phases in an epoch, the *initial testing phase* evaluates the error $e_{k,1}$ by simulating each of the m training cases and comparing the SNN's observed output decoded from Eq. (4.9) with the desired output computed from Eq. (4.5).

$$f(t_r) = \frac{\Sigma_{i=1,...,K} z_i(t_r)}{K} \tag{4.9}$$

where $z_i(t_r)$ is the total number of spikes of output neuron i within the time interval $[t - t_r, t]$. If K are the left motor neurons, Eq. (4.9) can calculate the left motor neurons' mean firing frequency or vice versa for right motor neurons.

If the error is below a criteria, the simulation will end. For each training case, the sensory stimuli is encoded as constant current injections, defined by Eq. (4.6)

and Eq. (4.7), to all input neurons for a duration of $t_e = 0.04$ seconds per case and with no pauses between cases. The time interval t_e is set to only be long enough for the signal to propagate through the SNN and be reliably decoded as an output. Then $e_{k,1}$ is obtained from Eq. (4.8).

The *initial training phase* of the epoch involves the delivery of a test training signal to the input neurons that is used to induce small synaptic weight changes in the SNN, which are then used to determine an extended training signal to be applied in the final training phase. The test training signal is given to a pair of input neurons i and j, selected from a list of all possible ordered pairs $\mathcal{N}$, such that $(i,\ j) \in \mathcal{N}$; $i, j = 1, ..., N;\ \ i \neq j$, where N denotes the number of input neurons. The training signals, denoted by s_i for the i^{th} neuron, are square pulse current inputs, such that,

$$s_i(t) = w \sum_{a=1}^{M} \left[H\left(t - p_{i,a} + \frac{\beta}{2}\right) - H\left(t - p_{i,a} - \frac{\beta}{2}\right) \right] \tag{4.10}$$

where $p_{i,a}$ represents the temporal center of the a^{th} square pulse delivered to the i^{th} input neuron, $M = 10$ is the number of square pulses in the test training signal over the duration of the initial training phase, $w = 7 \times 10^{-7}$ amperes is the amplitude of the pulses, and $\beta = 0.004$ seconds is the duration of each pulse, $H(\cdot)$ is the Heaviside function. The constants w and β are chosen such that each pulse will reliably induce the input neuron to spike once and only once. The pulse inputs are given every $t_p = 0.08$ seconds, and the pulses are offset slightly between the pair of input neurons $(i,\ j)$, such that $p_{j,a} = p_{i,a} + b_0$, where $b_0 = \pm 0.002$ seconds. This offset acts to induce small synaptic weight changes in the SNN through STDP.

The *final testing phase* of the epoch evaluates the error for a second time using the same procedure as in the initial testing phase. The error, $e_{k,2}$, computed from Eq. (4.8) will be different from $e_{k,1}$ since the synaptic weights of the SNN changed during the initial training phase and marginally changed during the initial testing

phase. The change in error, $\Delta e_k = e_{k,2} - e_{k,1}$, can then be computed and used to determine the optimal training signal.

The last phase of the epoch, the *final training phase*, uses the value of Δe_k to determine an optimal training signal that, when applied to the same pair of input neurons $(i,\ j)$, will modify the weights of the SNN to cause a decrease in error. As in the initial training phase, the training signal consists of square pulses computed from Eq. (4.10), but a new offset parameter, denoted by b_k, is used. The offset b_k is a function of the change in error, such that,

$$b_k = -\operatorname{sgn}(\Delta e_k) b_0 \tag{4.11}$$

The square pulses are then offset between the pair of input neurons $(i,\ j)$, such that $p_{j,a} = p_{i,a} + b_k$. This calculated offset ensures that the synaptic weight changes brought about by the training signal will decrease the error between the observed SNN output signal and the desired output. Also since the training must compensate for the weight changes from the other phases, and since the STDP mechanism is known to increase the weights at about double the rate that it decreases them [23], the optimal number of square pulses, M^*, is a function of b_k and Δe_k, and is given by,

$$M^* = \begin{cases} 2M, & b_0 > 0 \text{ and } \Delta e_k > 0 \\ \frac{1}{2}M, & b_0 < 0 \text{ and } \Delta e_k > 0 \\ M, & \Delta e_k < 0 \end{cases} \tag{4.12}$$

To summarize, the parameters to be determined that define the optimal training signal for each epoch are the training signal offset, b_k from Eq. (4.11), and the number of square pulses, M^* from Eq. (4.12). These parameters are functions of the change in error, Δe_k, which is a difference between the two evaluations of the error $e_{k,\ell}$, computed before and after an experimental training signal is applied. For each epoch, a new pair of neurons to receive the training signals is selected from the

ordered list $\mathcal{N}$, and the process is repeated until the error reduces to an acceptable value or stops decreasing. If $\Delta e_k < 0$, then $M^* = 10$. In Fig. 4.4, $w_{1,19}$ increases ten times according to the STDP rule. While if $\Delta e_k > 0, \quad b_0 > 0$, the number of training input pairs is $2M = 20$ in Fig. 4.4 bottom.

In Fig. 4.4, because the presynaptic neuron 1 fires before neuron 19 due to the temporal difference 0.002 (s) between the two square pulses, the synaptic weight $w_{1,19}$ increases. In contrast, neuron 5 fires after neuron 22 so that the synaptic weight $w_{5,22}$ decreases. The pseudo code is in Algorithm. 1.

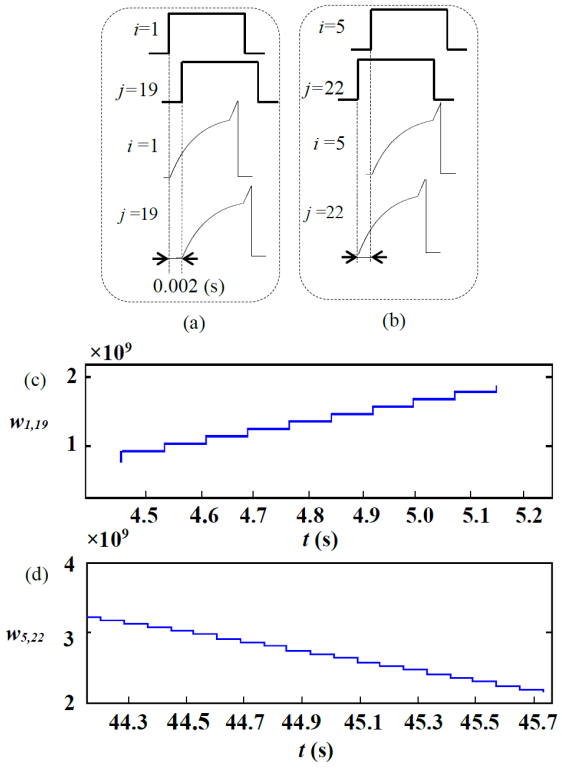


FIGURE 4.4: Action potentials of the pre and post-synaptic neurons and the weights change by training inputs. (a). Training square pulses for increasing the weight and response of the neurons. (b). Training square pulses for decreasing the weight and response of the neurons. (c). Weight increase caused by training stimuli during one training epoch. (d). Weight decrease caused by training stimuli during one training epoch.

Result: A trained SNN
Create an SNN;
while $e_{k,2}$>*criteria* **do**
  Test the SNN;
  calculate $e_{k,1}$;
  Pick neuron i, j;
  Generate s_i, s_j with temporal difference b_0;
  Stimulate i and j;
  Test the SNN;
  calculate $e_{k,2}$;
  if $e_{k,2}$>$e_{k,1}$ **then**
    Swap s_i, s_j;
  else
    Keep s_i, s_j;
  end
  Stimulate i and j;
  k=k+1;
end

Algorithm 1: Training algorithm pseudo code. T is the temperature. TD is the temperature decay factor.

4.1.2 Local Minimum

Mathematically, the *global minimum* is the smallest value of a given function in the entire domain while the *local minimum* is the smallest value in a given range in Fig. 4.5. Error functions with respect to the weights for multi-layer neural networks have many local minimums. One of the main weaknesses of the well known algorithm BackProp is that it can be trapped in a local minimum if the initial weights are not chosen well [65]. Therefore, comparing with BackProp and other gradient-based algorithms, genetic algorithms perform better in dealing with local minimum problems [4]. However, genetic algorithms are computational expensive and not biologically plausible. Our indirect perturbation algorithm also can settle into local minimum problem during training if the initial weights are not well chosen.

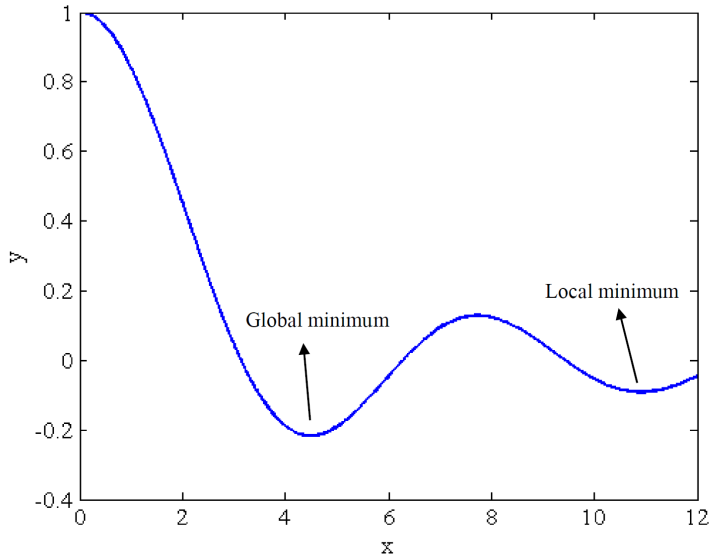
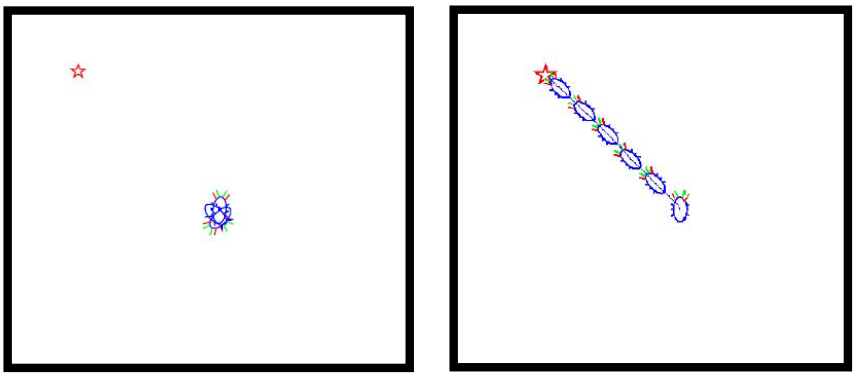


FIGURE 4.5: Global minimum and local minimum.

4.1.3 Application of Indirect Perturbation Algorithm on Virtual Insect Navigation

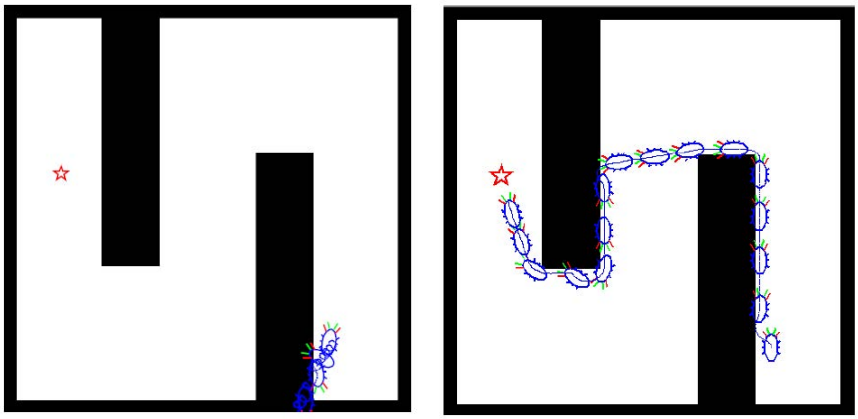
Insect Navigation

After learning, the virtual insect is placed in the three environments illustrated in Figs. 4.6-4.8, and compared to a naive insect controlled by the initial SNN model, with random synaptic strengths. It can be seen that, unlike the naive version, the trained SNN is capable of integrating information regarding the target location and terrain conditions and control the legs of the virtual insect such that it avoids elevated terrain, and reaches the target represented by a star. In particular, when placed in the obstacle-free environment, where only target information is relevant, the virtual insect navigates to the target using the path of shortest distance (Fig 4.6(b)), while the naive insect rotates in place (Fig. 4.6(a)).



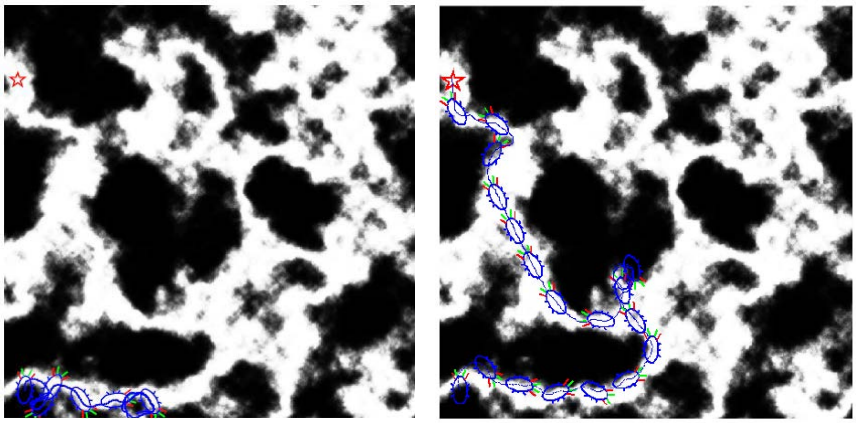
(a) (b)

FIGURE 4.6: Comparison of insect trajectories for naive (random synaptic strengths) (a) and trained (b) SNNs in an obstacle-free arena.



(a) (b)

FIGURE 4.7: Comparison of insect trajectories for naive (random synaptic strengths) (a) and trained (b) SNNs in an S-maze.



(a) (b)

FIGURE 4.8: Comparison of insect trajectories for naive (random synaptic strengths) (a) and trained (b) SNNs in a complex terrain of variable elevation.

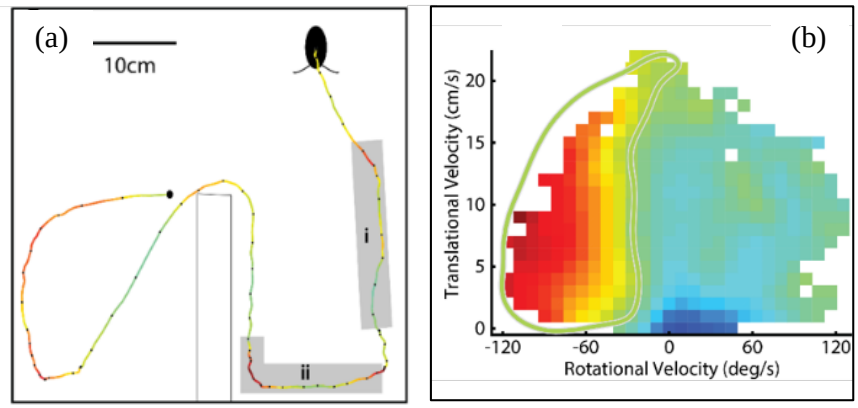


FIGURE 4.9: Track of live insect in the arena color coded based on the activity level of a CX unit (a). Color-coded neural (spike) activity as a function of translational and rotational velocity over entire track (b), where warmer colors indicate higher firing frequency (taken from [106]).

Similar to the arena used for biological experiments in Fig. 4.9, the S-Maze involves a target and two partial walls that the trained insect is capable of navigating around efficiently to find the target on the other side (Fig. 4.7(b)). Instead, the naive insect moves in circles locally, responding to tactile information, but is unable to

move past the first wall (Fig 4.7(a)). The irregular terrain characterized by hills and narrow channels represents a fairly complex landscape that would cause several path planning algorithms to remain stuck in dead ends or take long and inefficient detours. Instead, the trained SNN is capable to navigate efficiently and autonomously toward the goal, avoiding regions of high elevation and exploiting useful canyons between them (Fig. 4.8(b)). Although the naive insect can respond to stimuli from contact with obstacles, it is unable to reach the target (Fig. 4.8(a)).

When the simulation results are visualized in VRML (Fig. 4.10), the virtual insect also appears to display a realistic behavior (see insect video [176]). Besides validating the effectiveness of the proposed weight-free learning algorithm, these results also demonstrate that this spike-based formalism enables more direct and effective transfer of biological findings from animal experiments to synthetic platforms.

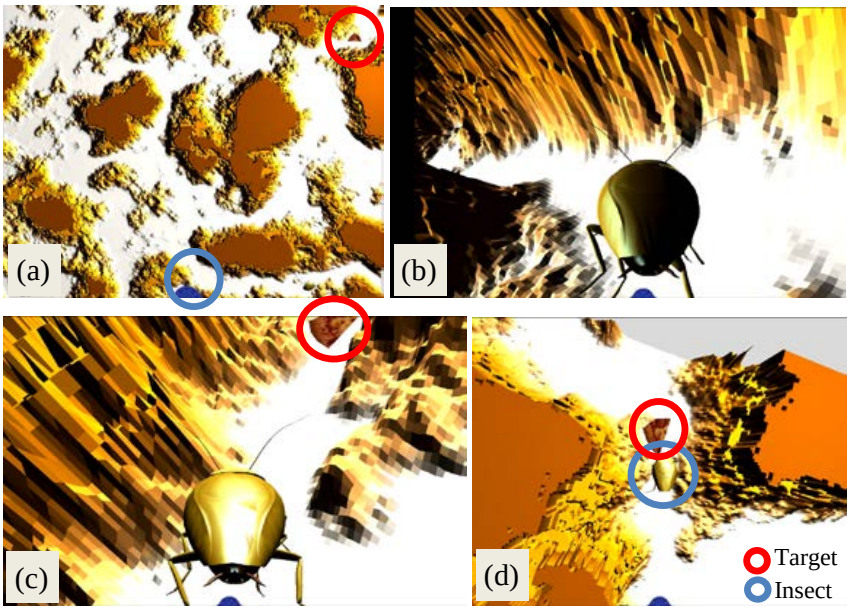


FIGURE 4.10: Insect behavior in a complex terrain (a), visualized in VRML, illustrating the ability of the trained SNN to navigate through narrow passages (b)-(c), ultimately reaching the target (d).

The scalability and robustness of the SNN weight-free learning algorithm are investigated by considering the effects of noisy sensory inputs, modeled by Eq. (4.6), on three SNN architectures comprised of 11, 14, 184 neurons, and 819 neurons, shown in Tables 4.1-4.4. Numerical tests show that, for noise-free sensory inputs, the weight-free learning algorithm can successfully train all four SNN architectures to properly control the virtual insect such that it can successfully and efficiently navigate all three environments. As an example, the root mean square error or training blue for the SNN with 819 neurons is plotted in Fig. 4.11. It can be seen that the error converges steadily to its minimum even for a large network size.

Table 4.1: SNN architecture with 11 neurons

Neurons	Excitatory	Inhibitory	Total
Input Sensor Neurons	8	0	8
Hidden Neurons	1	0	1
Output Motor Neurons	2	0	2

Table 4.2: SNN architecture with 14 neurons

Neurons	Excitatory	Inhibitory	Total
Input Sensor Neurons	6	2	8
Hidden Neurons	4	0	4
Output Motor Neurons	2	0	2

Table 4.3: SNN architecture with 184 neurons

Neurons	Excitatory	Inhibitory	Total
Input Sensor Neurons	49	15	64
Hidden Neurons	80	20	100
Output Motor Neurons	14	6	20

Table 4.4: SNN architecture with 819 neurons

Neurons	Excitatory	Inhibitory	Total
Input Sensor Neurons	124	20	144
Hidden Neurons	524	101	625
Output Motor Neurons	44	6	50

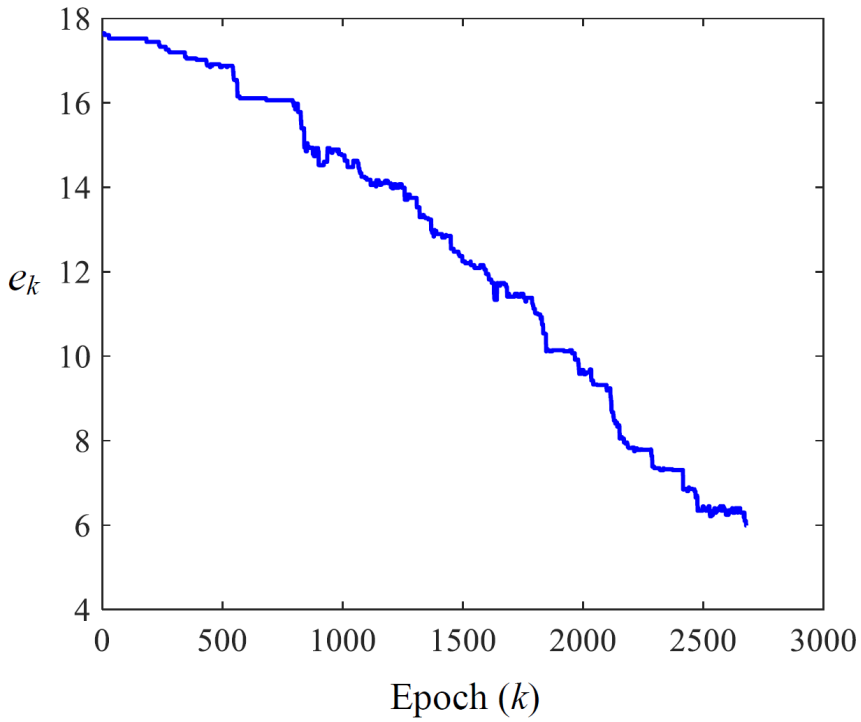


FIGURE 4.11: Training blue for SNN with 819 neurons.

In the presence of noise, the performance robustness improves as the size of the SNN is increased. Figure 4.12 shows the trajectories of the trained virtual insect controlled by the three SNNs in an obstacle-free environment, with and without sensor noise. It can be seen that the 11-neuron SNN is unable to navigate to the target in the presence of the sensor noise with $\nu = 0.5$. In contrast, the 14-neuron SNN is capable of controling the insect such that it reaches the target, but the insect first spins in a circle and only afterwards finds the shortest path. Comparing with 14-neuron SNN, the 184-neuron SNN shows a better performance, controlling the insect such that it corrects its orientation by turning around and then navigating to the target along the shortest path. The 819-neuron SNN shows the best performance, in which the noise almost has no impact on the robot traces. This insect distance from the target is plotted over time in Fig. 4.13 from the initial to a final time that

is chosen heuristically in each case. It can be seen that the largest (819-neuron) SNN performs better than all other SNNs, with or without sensor noise, and that in the presence of noise (with $\nu = 0.5$) the insect performance improves even more significantly with the SNN size.

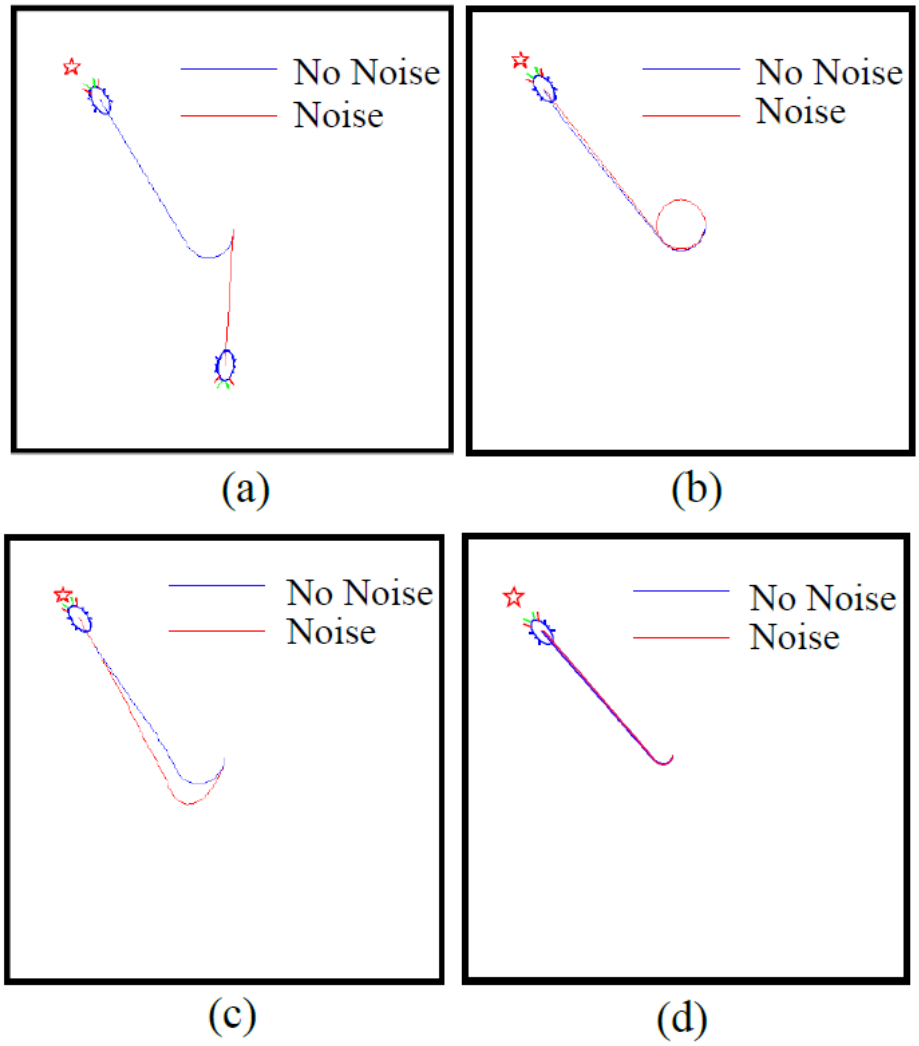


FIGURE 4.12: Trained insect trajectories with or without sensory input noise for SNN with 11 neurons (a), 14 neurons (b), 184 neurons (c) and 819 neurons (d) when $\nu = 0.5$.

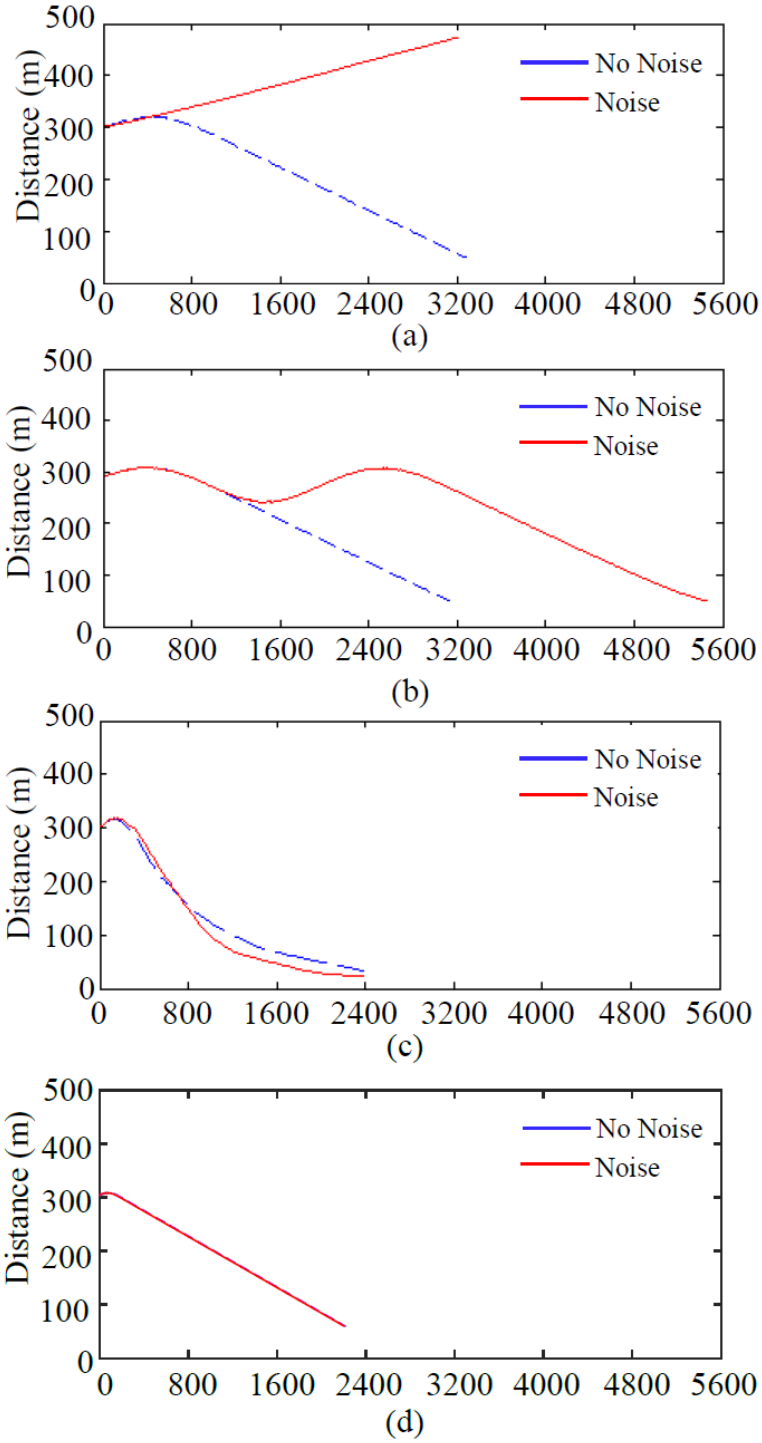


FIGURE 4.13: Time history of insect distance from the target in the presence of sensor noise, when $\nu = 0.5$, using the trained SNN with 11 neurons (a), 14 neurons (b), 184 neurons (c), and 819 neurons (d).

To further understand SNN robustness to noisy inputs, the insect performance is analyzed as a function of noise amplitude, $\nu \in [0,\ 1]$. By varying ν in the model of Eq. (4.6), random noise is produced with a magnitude up to $100 \cdot \nu$ (%) of the original sensor input ($g_{R,L}$ or $h_{R,L}$). Let success be defined as the ability to reach the target in a desired time window, in this case chosen as 5.6(s) based on the initial target distance and maximum insect speed. If the insect is unable to navigate to the target in this time window or leaves the arena, failure is declared.

By plotting the successes and failures as a function of noise amplitude (ν) in Fig. 4.14, the SNN robustness to noisy sensor inputs is established in the limit. It can be seen that, as expected, robustness increases with the size of the SNN, as the 819-neuron SNN is effective up to $\nu = 0.8$, while the 11-neuron SNN is only effective up to $\nu = 0.2$. This is attributed to the presence of a larger hidden layer capable of compensating for input errors. Surprisingly, the 14-neuron SNN can cope with noise up to $\nu = 0.5$, indicating that its limiting performance is acceptable up to a very significant noise amplitude. However, as shown in Figs. 4.12-4.13, the insect trajectories in this case are far less efficient than the trajectories obtained with the 819-neuron SNN. Thus, a large SNN can not only provide robust performance in the limit, but also robust *optimal* performance as may be desirable in many biological and engineering systems.

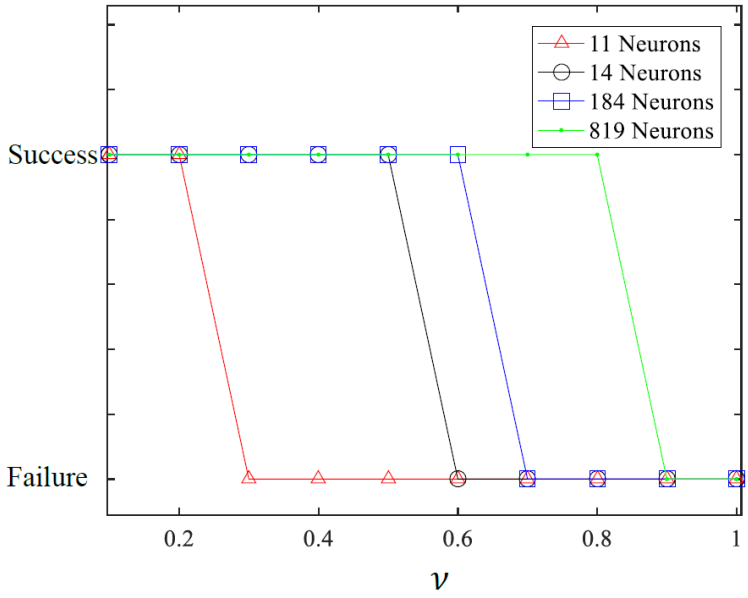


FIGURE 4.14: Trained SNN effectiveness as a function of sensor noise level and for different numbers of neurons.

The robustness of the weight-free learning algorithm to errors in stimulus delivery is tested by modifying the stimulus from the ideal square pulse function in Eq. (4.10), delivered only to neuron i, to a square pulse delivered to all neurons in a neighborhood of scale σ from i. By modeling the error in delivering the stimulus using a Gaussian distribution, the stimulus amplitude (or intensity) approaches w at neuron i, and decreases exponentially with the distance, $D(i,j)$, for any other neuron j nearby, or,

$$w_j = w \sum_i \exp\left[-\frac{D(i,j)}{2\sigma^2}\right] \tag{4.13}$$

where the variance is chosen by the user. In this paper, the value $\sigma = 0.8$ is found to adequately represent the error found in neuronal cultures, causing multiple nearby neurons to fire, as shown in Fig. 4.15 for a neuron i with coordinates $(X, Y) = (2, 2)$. When the indirect perturbation algorithm is implemented with the stimulus error in Eq. (4.13), the network performance can still be optimized reliably, as shown by the

training blue in Fig. 4.16.

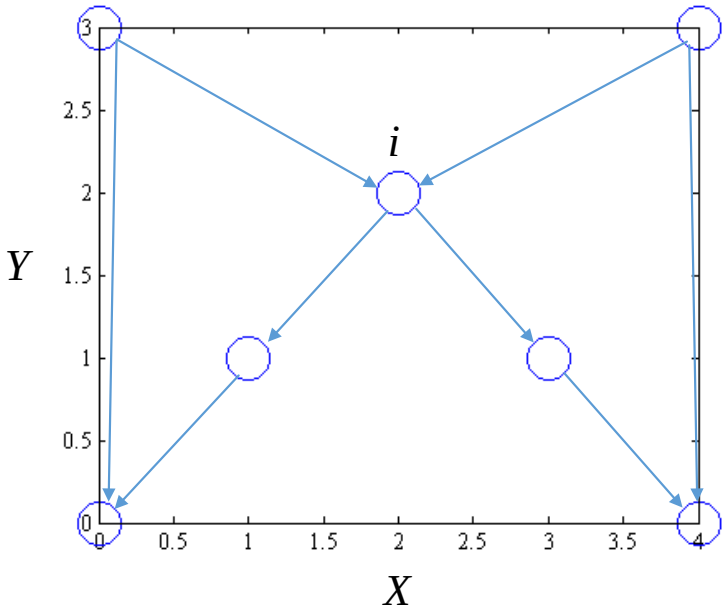


FIGURE 4.15: Seven neurons influenced by stimulus with Gaussian error in Eqn. (4.13) when training pulse is delivered to neuron i at $(X, Y) = (2, 2)$.

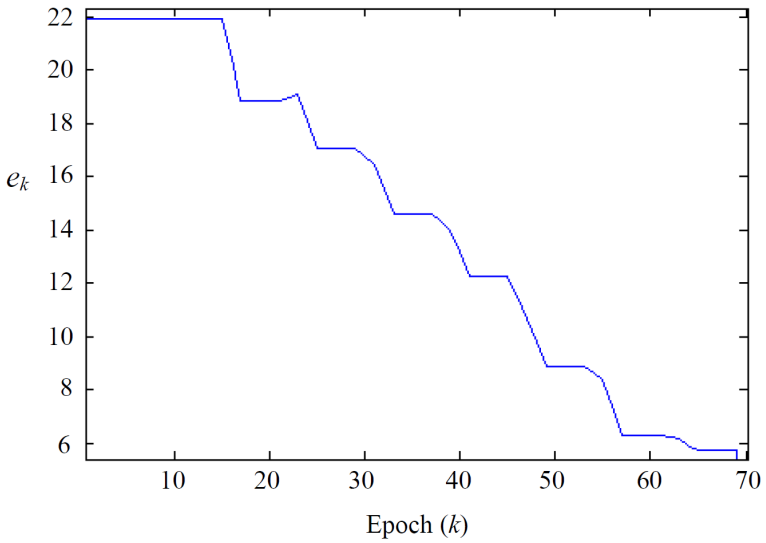


FIGURE 4.16: Training blue in the presence of training stimulus error, for $\sigma = 0.8$ with the network in Fig. 4.15.

4.1.4 Application of Indirect Perturbation Algorithm on Indoor Robot Navigation using Optitrack

Neurorobotics, as a combination of neuroscience, robotics, and artificial intelligence, is the research field of embodied autonomous neural systems [87]. Neural systems have bio-inspired algorithms, computational model of neural networks like spiking neural networks and biological neural networks in *vivo* or *vitro*. These networks can be implemented on machines with mechanical output or other output signals including robots, prosthetic devices. Neurorobotics tools help to reverse engineer cognitive function and its architecture in the brain.

Neuroscience tends to study intelligent behaviors by studying the possible theory behind those intelligent behaviors using experiment and analysis. While artificial intelligence believes those problems can be solved in other ways other than mimic human or other animal's behavior. Neurorobotics combines these to field by testing those biologically inspired theories with a physical implementation. The results of those experiments can provide evidence to reject or accept those proposed theories. Neurorobots include three major classes including the study of motor control, memory, action selection, and perception.

The neural network used for controlling the robot with *Optitrack* is trained by the indirect perturbation algorithm in Sec. 1.1. It is an off-line training, that is ran through the SNN simulator Brian 2. In general, the algorithm includes two training phases and two testing phases. The first test phase is used to measure the initial root mean square error $e_{k,1}$. After the first test phase, two neurons within the network are selected and stimulated through square pulses, which have temporal difference b_0. In the second testing phase, $e_{k,2}$ is calculated and used to calculate the error change $\Delta e_k = e_{k,2} - e_{k,1}$. If $\Delta e_k > 0$, two square pulses with the opposite temporal differences of previous one are used to stimulated those two selected neurons. After training, the network weights are saved for neurorobot experiments.

The neurorobotics experiment is done in Duke Zavlanos Lab using robot built based on irobot create^{TM} shown in Fig. 4.19. Robot Operating System (ROS) is installed on the computer embedded on iRobot for controlling the robot by higher level of command comparing with irobot's own commands. The lab is equipped with OptiTrack, which is a 6 DoF object tracking system that can locate the positions of the markers precisely in millimeters. Therefore, there are 24 cameras on the ceilings of the lab for covering the locations of a square areas around 3*3 m^2. The information captured by the cameras are sent to a PC with installed software *Motive* for receiving information through TCP/IP protocol. The SNN is also simulated on that PC, which has 8 core Intel Xeon (R) CPU 2.5Hz and CUDA capable GPU *NVIDIA Quadro K4000* for running GPU accelerated SNN simulator CarSim. The outputs of the SNN are sent to the robot as left wheel and right wheel velocity command via TCP/IP protocol. The input information update for the SNN is 10 Hz. In other words, each short SNN simulation for update the new output command takes 100 ms. Two markers on the robot were picked and adjusted by rotating the platform, where they were pasted so that they are approximately symmetric according to the axis between two wheels.

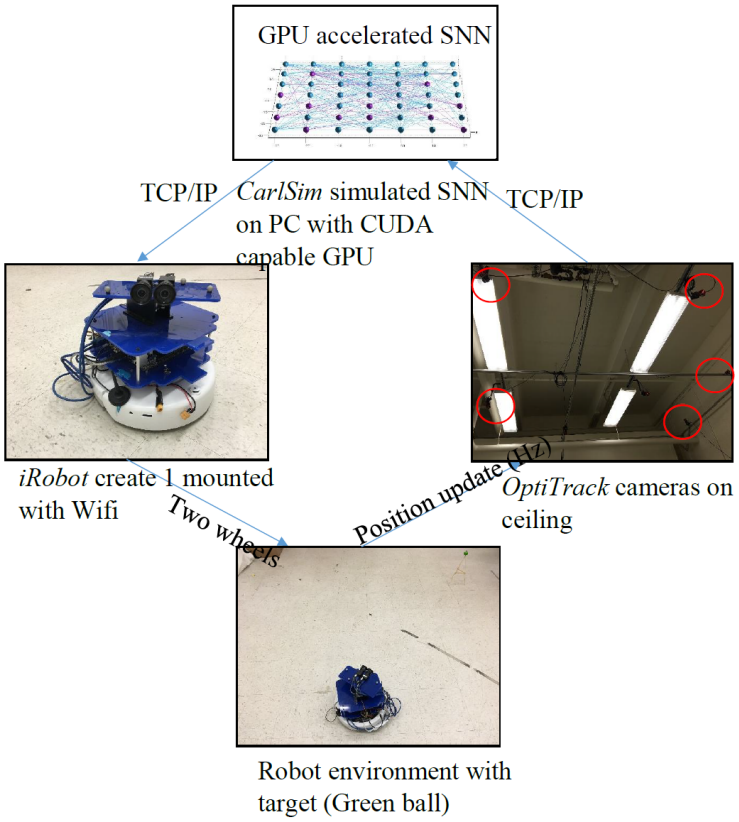


FIGURE 4.17: The robot experiment communication system.

The SNN simulator CarlSim is installed and running on *Microsoft Visual Studio* 2012. The SNN simulated is a recurrent NN with 49 neurons in Fig. 4.18.

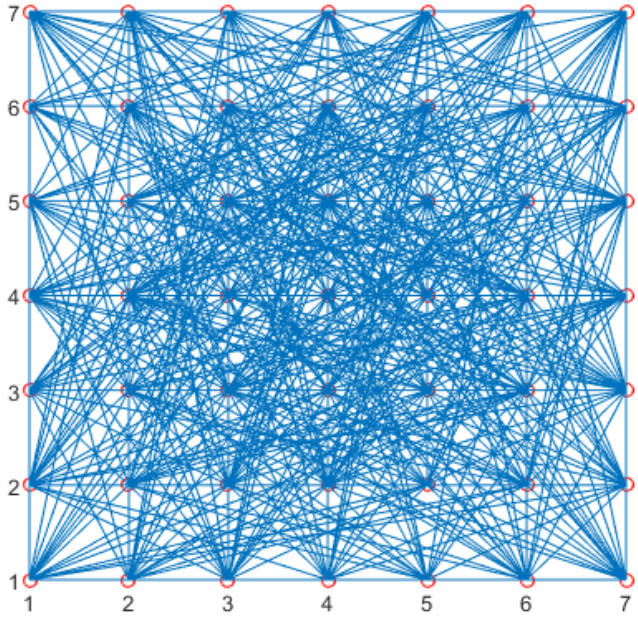


FIGURE 4.18: The SNN with 49 neurons and random recurrent connections. The square shape is only for better illustration of those connections.

Neurons in Fig. 4.18 are modeled by Izhikevich neuron. Their weights are trained by using Brian 2. The trained neural network's weights and other data are saved in a txt file and imported by CarlSim in C++. Real time information coming from the cameras are used to calculating the Euclidean distance between the two markers on the robot and the target location.

$$d_l = \sqrt{(x_l - x_{target})^2 + (y_l - y_{target})^2} d_r = \sqrt{(x_r - x_{target})^2 + (y_r - y_{target})^2} \quad (4.14)$$

where d_l, d_r are distances between the left, right target sensor and the target, (x_l, y_l) is the position of the left marker, (x_r, y_r) is the position of the right marker, (x_{target}, y_{target}) is the position of the target.

Results of Indoor Robot Experiment

First, the robot is tested on an environment without obstacles. The robot is given different initial positions and orientations so that the traces include both turning left and right. To avoid collision, the robot inputs are set to zero when the distance d_l and d_r are less than a criteria d_c. Fig. 4.19 shows those traces, which all successfully reach the target plotted as blue circle.

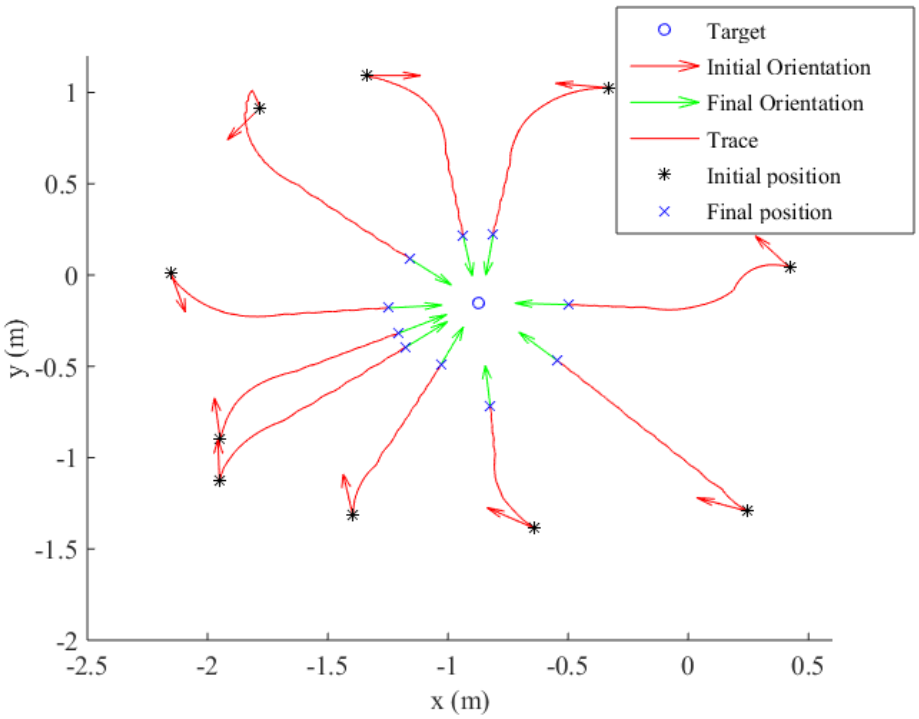


FIGURE 4.19: Robot traces with different initial positions and orientations.

For better illustration of the spikes during the experiment, Fig. 4.20 shows the responses of those 49 neurons during certain periods of the experiment corresponding to the robot's positions in A, B, C. At initial position, neuron 35-37, 42-44 fire with high frequencies because they receive the right target sensor values, which are much larger than the left. In the raster plot of position near B, the high frequency firing neurons switch to neuron 39-41, 47-49 because it turns slightly over. At position

near C, the robot continuously adjusts its orientation causing the neuron spikes to oscillate.

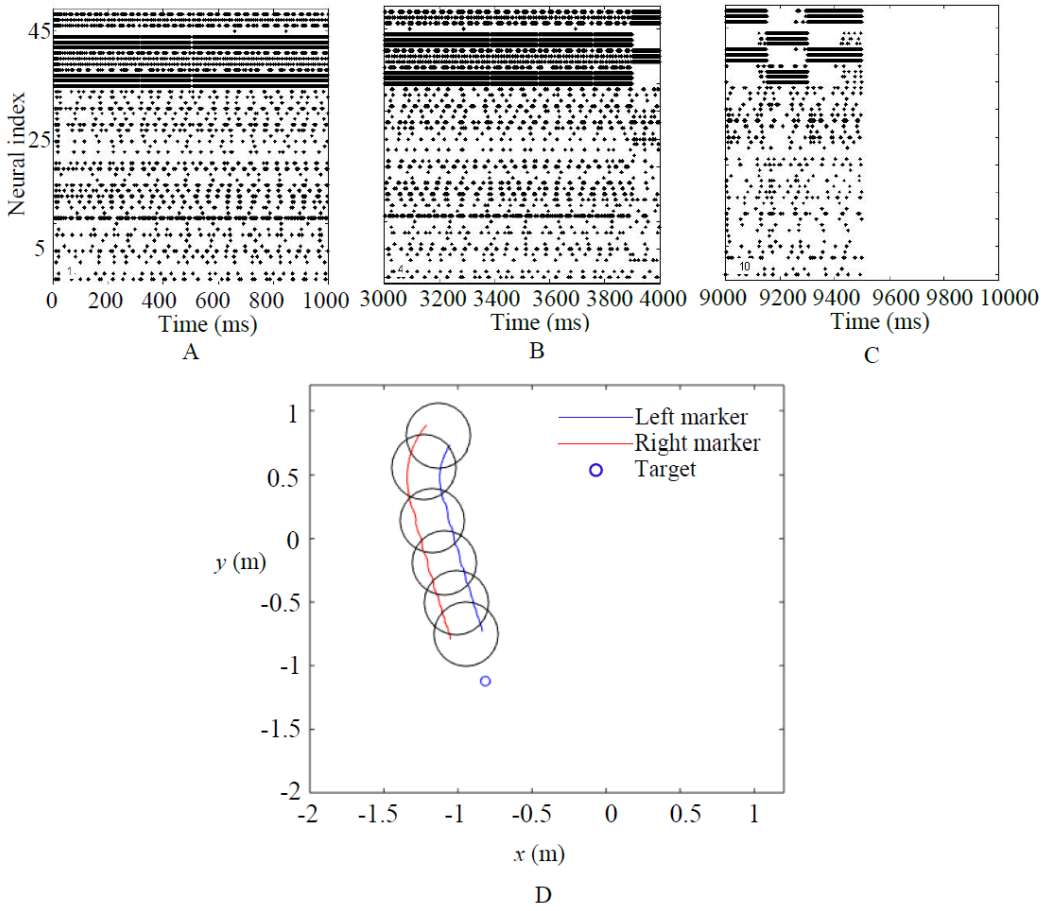


FIGURE 4.20: Spikes of the SNN during robot's turn and reaching the target. (A). Spikes of the SNN around position A. (B). Spikes of the SNN around the turning point at position B. (C). Spikes of the SNN around the final point at position C. (D). Robot trace during the simulation.

In simulation or in experiment performed indoors, the assumptions and conditions are more static and certain than outdoor environment. For example, OptiTrack can provide millimeters localization for the robots in the lab, which is impossible to provided in outdoor environment. GPS in outdoor environment have only meters precision. Therefore, the robustness of the SNN under sensor noise is very important

for implementation of the robot in outdoor environment. In order to test the trained SNN's performance with sensor noise, noise defined as,

$$s_{noise} = (2 * rand - 1) * \alpha * s \tag{4.15}$$

is added when s_{noise} is the sensor noise, $rand$ is the rand number between 0 and 1, α is a constant, s is the original sensor value. Given $\alpha \in [0.1, 0.9]$, the robot traces with approximately the same initial position and orientation are plotted in Fig. 4.21. The robot successfully reaches the target with sensor noise from 10% to 40%. The larger the noise, the longer trace it will take for the robot to reach the target. A robot with 50% sensor noise may reach the target if the experimental environment is sufficiently large. The robot cannot navigate with the sensor noise larger than 50%.

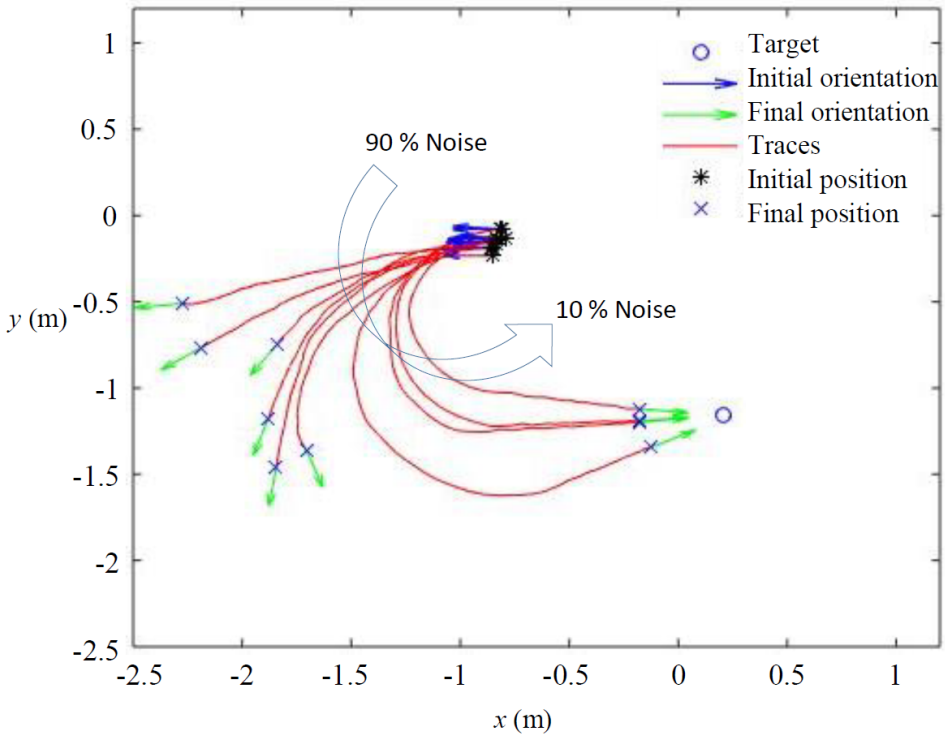


FIGURE 4.21: Spikes of the SNN during robot's turn and reaching the target. (A). Spikes of the SNN around position A. (B). Spikes of the SNN around the turning point at position B. (C). Spikes of the SNN around the final point at position C. (D). Robot trace during the simulation.

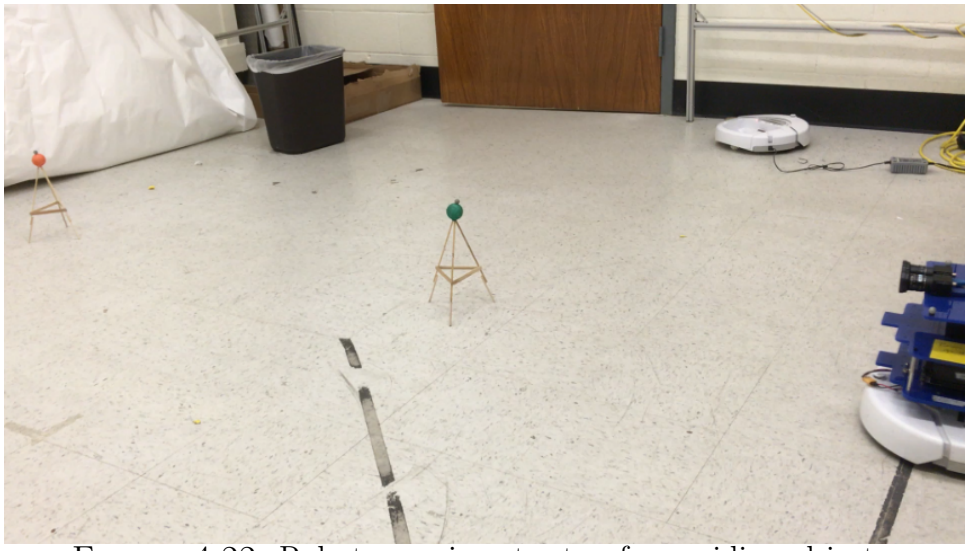


FIGURE 4.22: Robot experiment setup for avoiding objects.

To show the robot's ability to avoid obstacles, one extra marker is placed between the robot and its final target. Fig. 4.22 shows the experimental set up. The marker is modeled with a terrain sensor value defined by,

$$s_{tr} = \frac{D}{d_{tr}} \tag{4.16}$$

$$d_{tr} = \sqrt{(x - x_{tr})^2 + (y - y_{tr})^2} \tag{4.17}$$

The robot trace is shown in Fig. 4.23. The robot successfully avoids the obstacle and reaches its target. Spikes during four important periods of the experiment are plotted in Fig. 4.24. During the SNN simulation around position A, the robot is far from the obstacle shown as red circle. Therefore, only neurons receiving target sensor values fire with high frequencies. At position B, the robot is within the region covered by the obstacle sensor values. Neuron 4-6, 11-13 fire with high frequencies. At position C, the robot runs out of the obstacle region, the frequencies of neuron 4-6, 11-13 decrease. When the robot reaches near the target at position D, the firing neurons oscillate.

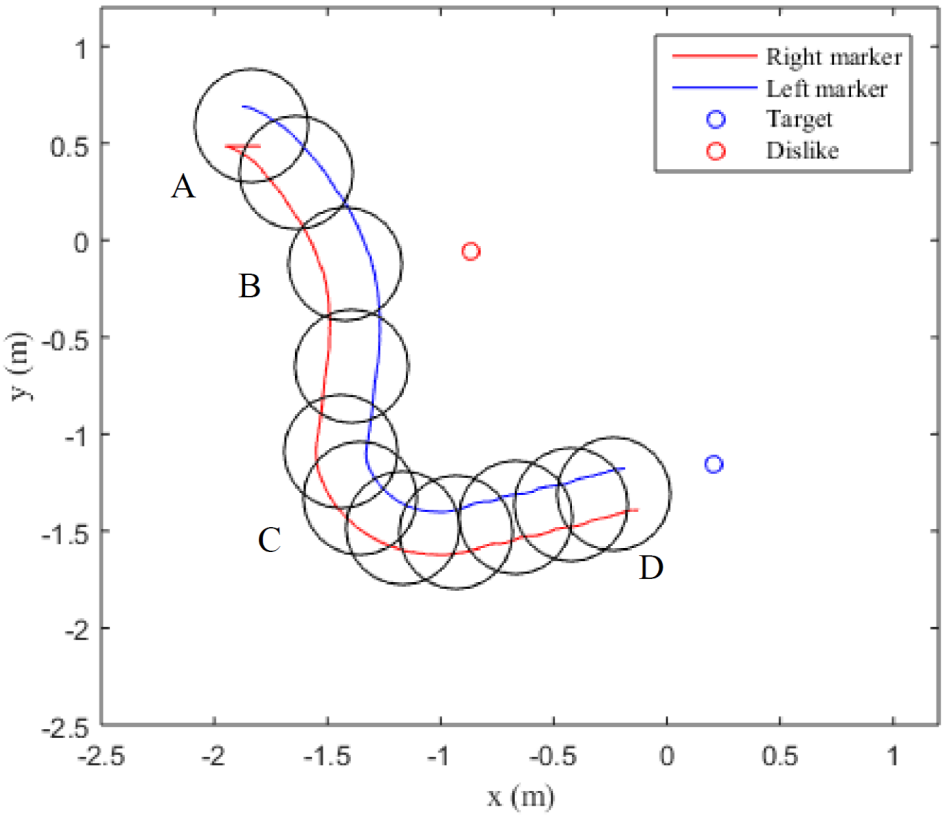


FIGURE 4.23: Spikes of the SNN during robot's turn and reaching the target. (A). Spikes of the SNN around position A. (B). Spikes of the SNN around the turning point at position B. (C). Spikes of the SNN around the final point at position C. (D). Robot trace during the simulation.

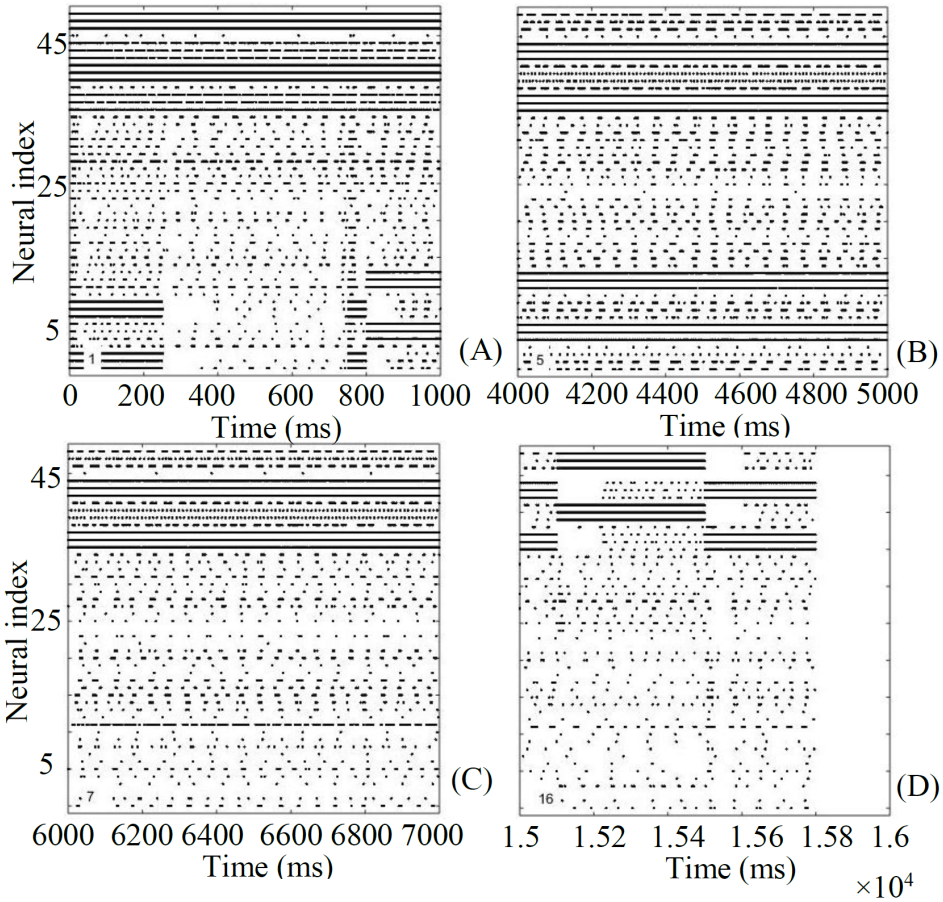


FIGURE 4.24: Spikes of the SNN during robot's turn and reaching the target. (A). Spikes of the SNN around position A. (B). Spikes of the SNN around the turning point at position B. (C). Spikes of the SNN around the final point at position C. (D). Robot trace during the simulation.

4.1.5 Discussion

This chapter presents an indirect training algorithm capable of manipulating plasticity at multiple scales of neural circuits. The advantage of a indirect algorithm is that plasticity is altered by controlling cell activity and, thus, it is potentially realizable in live biological neuronal networks via optogenetics or electrical stimulation. The idea of training neural networks by controlling cell activity, rather than by controlling the synaptic weights according to a learning rule, was recently demonstrated on small-scale SNNs [177]. This idea is motivated by techniques, such as optogenetics, that

are having a significant impact in the neuroscience field by delivering optical firing control with the precision and spatiotemporal resolution required for investigating information processing and plasticity in biological brains.

Even in the simplest organisms, however, brain circuits are characterized by hundreds of neurons responsible for integrating diverse stimuli and for controlling multiple functionalities. Because of their size, these large neuronal structures are believed to provide robustness and reconfigurability, characteristics that are also desirable in neuromorphic circuits and engineering applications of computational SNNs. This chapter presents a new perturbative learning algorithm for scaling the weight-free paradigm up to networks with over eight-hundred neurons.

The results in this chapter show that the indirect perturbation method is feasible for training SNNs on a larger scale than previously shown in the literature, and without any knowledge of their connectivity or synaptic strengths. The spike-based method by perturbation has a good level of accuracy and reliability in the chosen application of virtual insect control and neurorobotic navigation. The virtual insect application is motivated by experimental studies in biology which have recently revealed that changes in the activity (e.g. firing rates) of individual units immediately precede changes in the firing rates of motoneurons responsible for locomotory behaviors such as walking speed, turning, and climbing [132, 133, 134]. The same wires used for recording cell activity can be used to stimulate the brain region to evoke altered behavior and plasticity. Hence, the indirect perturbation learning algorithm is demonstrated on a virtual simulation of the aforementioned insect experiments to demonstrate its effectiveness at inducing higher-level learning, as well as to illustrate how it might some day be used for controlling plasticity in experiments on living insects. The simulation results show that the trained SNN sensorimotor controller is capable of integrating different sensory stimuli and accomplish desired behavioral goals efficiently and reliably. As expected, the robustness of the trained SNN is shown

to improve significantly with network size, providing near optimal performance even in the presence of large sources of sensor noise.

The neurorobot experiment shows that the indirect perturbation algorithm can also train networks modeled by the Izhikevich model to control real robots for navigate in real world environments with more unexpected accidents and sensory noises. The results show that the robot can search in an unknown environment and find its target in a real time speed.

4.2 Indirect Stochastic Gradient Descent Algorithm and Application

4.2.1 Algorithm Description

The Indirect Perturbation algorithm described in Sec. 4.1 can be classified as a hill climbing algorithm [148]. It is guaranteed to converge to a local minimum with a small enough step size to avoid oscillation. Gradient Descent can converge faster than Indirect Perturbation with fewer requirements on the step size. For problems requiring large data sets such as ImageNet, MNIST and CIFAR, the training of traditional neural networks (e.g., Convolutional Neural Network (CNN), take days or even weeks to train [89]. Therefore stochastic gradient approaches are often used in which a subset of data is randomly chosen from the whole data set and calculate the gradient of error with respect to the weights. After all the subsets are used, the algorithm will randomize the data again and continue to pick subsets. In our case, the data set for training is not particularly large. The time to implement the algorithm can be significant due to the indirect nature in which square current pulses are used to modify the weights. In the indirect method described below, Stochastic Gradient Descent is used but applied to randomly selected subset of weights rather than data to calculate the gradient in,

$$\frac{\partial E}{w_{ij}} = \frac{\Delta E}{\Delta_{perturb} w_{ij}} + O(\Delta_{perturb} w_{ij}) \tag{4.18}$$

If the perturbation $\Delta_{perturb} w_{ij}$ is small enough, the weight update rule can be,

$$\Delta w_{ij} = -\eta \cdot \frac{\Delta E}{\Delta_{perturb} w_{ij}} \tag{4.19}$$

The Indirect Stochastic Gradient Descent Algorithm can be written as a pseudocode in Algorithm. 1. The algorithm uses a randomly generated SNN. Then randomize possible pairs are grouped into 100 subsets. The batch number is initialized to be zero. A criteria is initialized to terminate the train if it is satisfied. During each while loop, an error is generated by testing all cases. Then one batch of i, j is selected to be perturbed indirectly and used to calculate the corresponding gradients of error with respect to the perturbation. After all the gradients of one batch are calculated, corresponding square pulses are generated to stimulate those i, j. The number of square pulses for each i, j are determined by the gradients calculated.

Data: Initialize the training cases
Result: A trained SNN
initialization;
Randomize pairs into subsets W_{batch};
$batch = 0$;
while $E > criteria$ **do**
  Run six different cases;
  Calculate E;
  if $batch > 100$ **then**
    randomize pairs into subsets W_{batch}
  end
  for i *and* j, $i, j \in W_{batch}$ **do**
    Stimulate i and j with time difference δt;
    Run six different cases;
    Calculate $E^{ij}_{perturb}$;
    Calculate gradient $G_{ij} = \frac{E^{ij}_{perturb} - E}{A^+ \exp \frac{\delta t}{\tau_+}}$
  end
  for i *and* j, $i, j \in W_{batch}$ **do**
    Calculate $\delta w^*_{ij} = -\eta * G_{ij}$;
    Calculate number of spike pairs needed N_{spair};
    for $i = 1 : N_{spair}$ **do**
      Stimulate i and j;
    end
  end
  $batch+ = 1$;
end

Algorithm 2: Indirect Stochastic Gradient Descent Algorithm

4.2.2 Performance of Indirect SGD Vs. Indirect Perturbation Algorithm

In order to better show and analyze the performance of Indirect SGD method described in Sec. 2.1, networks with the same size and connection properties are trained using both indirect SGD algorithm and indirect perturbation algorithm. Fig. 4.25 shows the error change during training using indirect SGD and indirect perturbation algorithm.

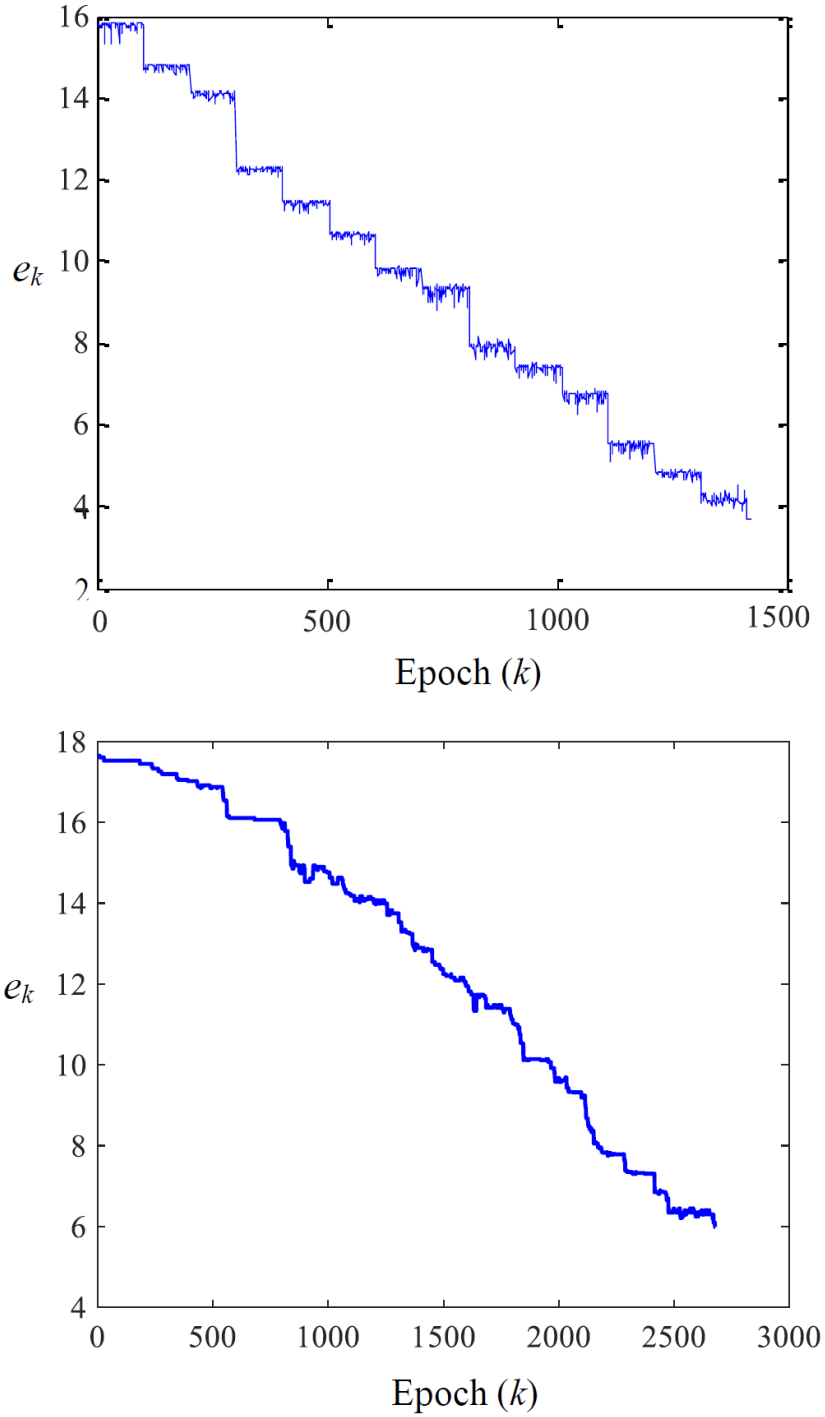


FIGURE 4.25: Error during training. (a). Error during training using Indirect Perturbation Algorithm. (b). Error during training using Indirect SGD.

The results show that the indirect SGD can train large size networks with better performance than the indirect perturbation algorithm. To further analyze the performance differences of the networks trained by these two algorithms, they are both used to control a virtual insect to navigate on an blank terrain. Their performance are compared under different level of sensor noise defined by Eqn. 6.5.

Fig. 4.26 shows the insect traces controlled by one indirect SGD trained network and one indirect perturbation trained network for sensor noise with $p = 0.9$. The indirect SGD trained network performs better by controlling the insect to reach the target by moving along a shorter path. This can be further seen in Fig. 4.27, which plots the distances from the insects to the target during the test. The indirect SGD controlled insects uses shorter path than the indirect perturbation metho.

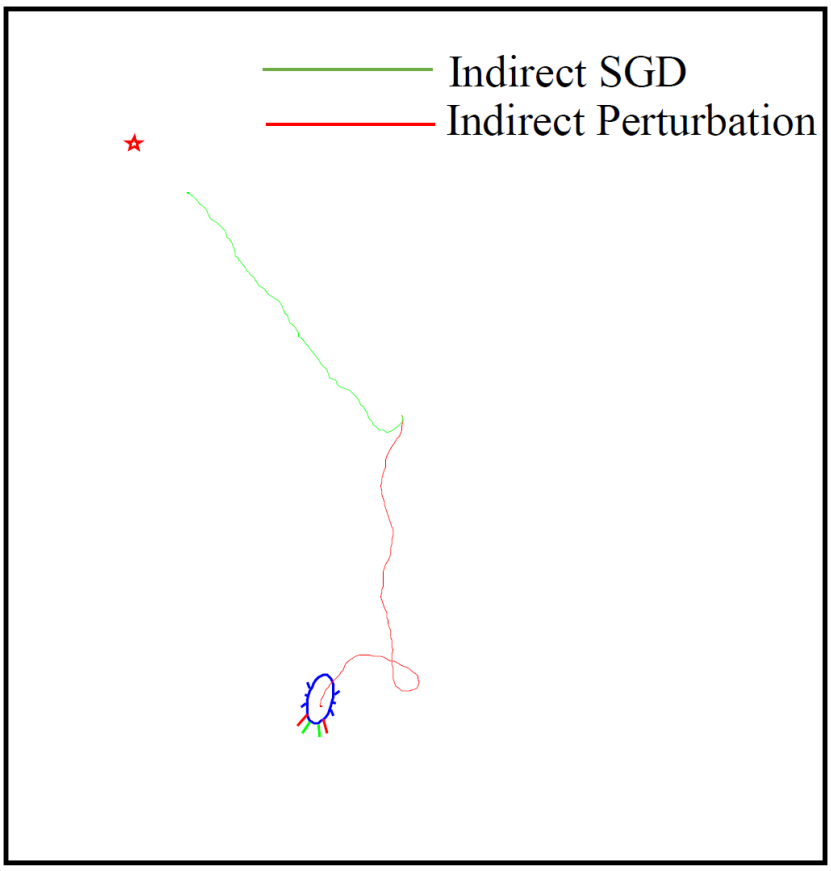


FIGURE 4.26: Traces of the insects trained by (a). indirect SGD (green) and (b). indirect perturbation (red) under sensor noise $p = 0.9$.

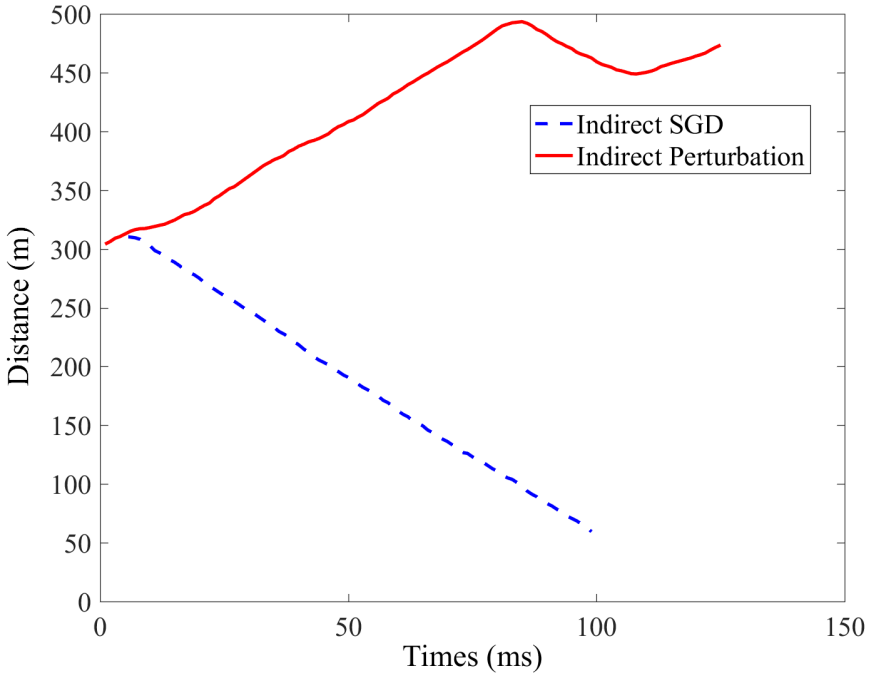


FIGURE 4.27: The distance from the insects to the target of insect controlled by (a). indirect SGD (blue) and (b). indirect perturbation (red).

In addition, the insect navigation performance is tested under different level of noise. As we can see in Fig. 4.28, the network trained by indirect SGD can tolerate large sensor noise than the network trained by indirect perturbation algorithm.

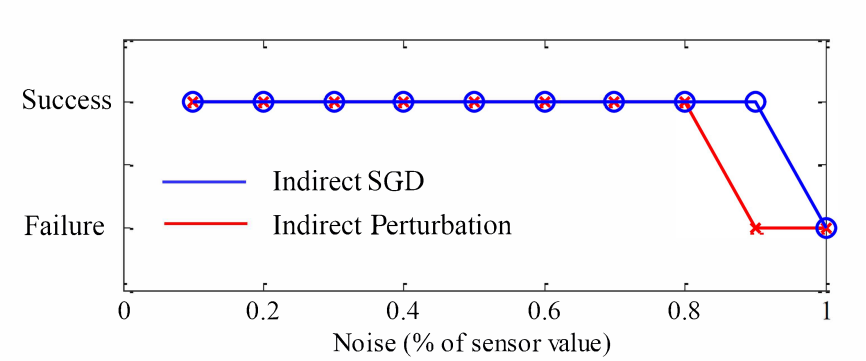


FIGURE 4.28: Performance of the insects under different level of sensory noises trained by (a). indirect SGD (blue) and (b). indirect perturbation (red).

4.2.3 Discussion

The indirect SGD algorithm was developed because the standard indirect perturbation algorithm can take a long time to train and sometimes will not converge to a good optimal value due to the oscillation. The results show that the indirect SGD algorithm can train a network to lower error comparing with indirect perturbation algorithm. In addition, indirect SGD algorithm takes shorter time to run comparing with indirect perturbation. The trained networks are tested on controlling virtual insects to navigate on a blank terrain under large sensor noise. For the indirect perturbation method, the trace of the insect controlled by an SNN is generally longer. In contrast, the indirect SGD trained network can drive the insect to the target in a shorter path. The indirect SGD trained network can also tolerate large noise than the indirect perturbation trained network.

The indirect SGD algorithm's performance depends on several parameters. First, the scale of the initial weight change for measuring the gradient is very sensitive. An inappropriate scale can cause either vanishing gradients or exploding gradients. This problem occurs for often in rate coding problems where a small perturbation of the weight cannot cause the output to change. Therefore, the perturbation should be larger than the perturbation than that used in traditional SGD algorithm. Secondly, the setting of the desired output values can also affects of the final error. If the desired output values are too large, the weights can increase such that stimulating only two neurons can cause multiple neurons to fire. In this case, reducing the scale of the desired firing frequency is needed.

The primary limitation of both the indirect SGD and indirect perturbation method is that they require that only two neurons fire during each training epoch. If many neurons fire during training epoch, unexpected weight changes may increase the error during each training epoch. Therefore, initialization of the weights have to be low

enough so that stimulating two neurons will not cause other postsynaptic neurons to fire. Because of the rate coding, during each training epoch, some gradients might be zero. This can be solved through increasing the testing period length or increasing the weight perturbation value for calculating the gradients.

4.3 Indirect Training with Supervised Teaching Signals and Neurorobot Applications

4.3.1 Algorithm Description

Some algorithms use teaching signals that are directly given to the output neurons to facilitate the weights to change based on STDP so that the output neurons can fire in desired patterns. [17] designed the first supervised STDP-like algorithm that can train an Spiking Neural Network to classify Latex symbols with high accuracy. The paper uses simple LIF neurons with constant leak and Dirac pulse modeled synaptic current. Two main weakness of their work are the oversimplified models and hand tuned learning parameters. Beyeler [7] used the same STDP-like learning mechanism, but with more biological neural and synapse models including Izhikevich neuron models and conductance based synapses. Even though the paper uses a multilayer neural network structure, the learning actually only happens in the final layer. Tempotron-rule is another supervised learning algorithm that can be used to train SNNs to fire or not fire. Therefore it is not well suited to solve for function approximation tasks [69]. As a result, another indirect training algorithm that can analytically determine the training signals and can train biological plausible neural network models is needed.

In the chapter I present an indirect algorithm which uses supervised teaching signals to train a two layer network with 3800 neurons in Fig. 4.29. The first layer includes 3600 neurons with all-to-all feed forward connection to the second layer with 200 neurons. The dynamics of synapses follow the STDP rule described in Chapter.

2. The neurons are modeled by Izhikevich model described in Chapter. 2. One training epoch has two phases: a testing phase and a training phase. During the testing phase, only the inputs are given to the neural network. So the error will be measured. Before the training phase, the desired training stimuli is calculated by Eq. 4.21.

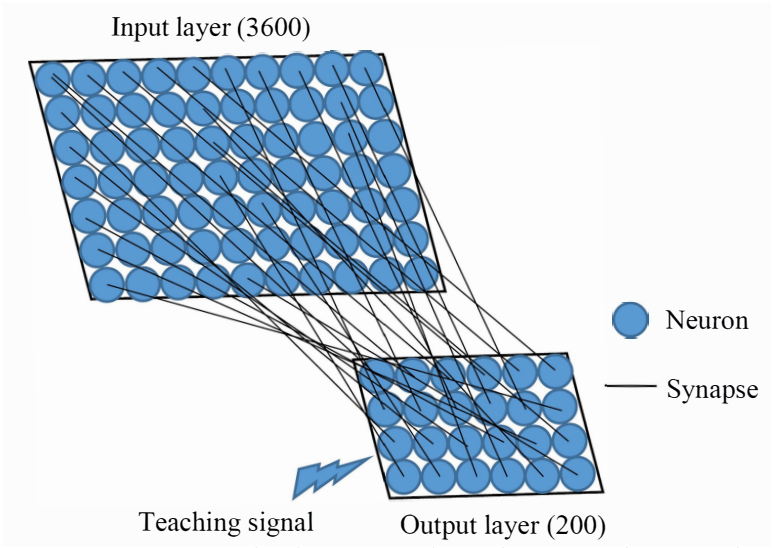


FIGURE 4.29: The largest indirectly trained network

The desired controller is supposed to control the output neurons to fire at a constant average firing frequency by adjusting the teaching stimuli correspondingly so that the weights will update in a controllable rate. Therefore, there is a training signal adjustment session before the training session, during which the STDP is turned off. During the training signal adjustment session, the stimulus is updated by,

$$S_k = S_{k-1} + \alpha * (f_t^* - f_t) \tag{4.20}$$

where S_k, S_{k-1} are the teaching signal's amplitude during $k_{th}, k-1_{th}$ epoch, α is the adjusting rate, f_t^* is the desired firing rate of the output neurons during training, f_t is the actual firing rate of the output neurons during training. After the desired

training signal is measured during the training signal adjustment session, the STDP is activated; and the inputs and training signals are given to the SNN to update the weights for a certain time period T_{train}. Then the performance of the network is measured during a testing period by,

$$e(t) = f^*(t) - f(t) \tag{4.21}$$

where $f^*(t)$ is the desired average firing frequency of the output neurons during testing period, $f(t)$ is the actual average firing frequency during testing period. The training takes many training epochs based on the network size and complexity of the problem to be solved. The pseudo code Alg. 4 can be found in Appendix. A.

4.3.2 Neurorobotic Navigation using Large SNNs and Embedded Cameras

In order to process images, the network size has to be much larger than the network used to control the robot based on *Optitrack*. Therefore, the number of synapses can be 100 times more than in the smaller network, which exponentially increases the training time if only two neurons are stimulated during each training epoch. Consequently, the indirect training with supervised teaching signals method was developed (Sec. 3.1) for this type of problem. A two layer feedforward network was used to test the algorithm . The training is composed of both testing epoch and training epoch. The network is tested by giving the input layer testing data and then evaluate the root mean square error of the output and the desired output. If the output firing rate is too small, the input layer is stimulated and the output layer is stimulated to make the output fire at a certain frequency f^*_{tinc}. This will cause the corresponding weights among those stimulated regions to be strengthened. When output firing rate is too high, the input layer is stimulated and the output layer is stimulated to fire in f^*_{tdec} so that the corresponding weights will be weaken. The detail of this method can be found in Sec. 3.1.

The network was trained to process the input image taken from the embedded camera and output a commend to control the robot's wheel speeds. A trained network can control a robot to reach its target by perceiving the image from the embedded camera and output a desired movement. Half of the output neurons are supposed to control the left wheel while the other half are used to control the right wheel by calculating the average firing frequency during a testing epoch. Therefore, the motor speed can be defined as,

$$V_i = \sum_{j \in O} \gamma \frac{f_j}{N} \tag{4.22}$$

where V_i is the velocity of motor i, j is the index of the output neurons, O is the index set, γ is a constant scalar, f_j is the firing frequency of neuron j, N is the total number of neurons in that index set.

Training Large Size Neural Network

The network shown in Fig. 4.29 is initialized with random weights that can be seen in Fig. 4.30. During training, regions of neurons are stimulated during each training epoch. Therefore a large amount of weights are updated during each training epoch so that this indirect training method is the most efficient one among those described before. The weights during training can be seen in Fig. 4.31.

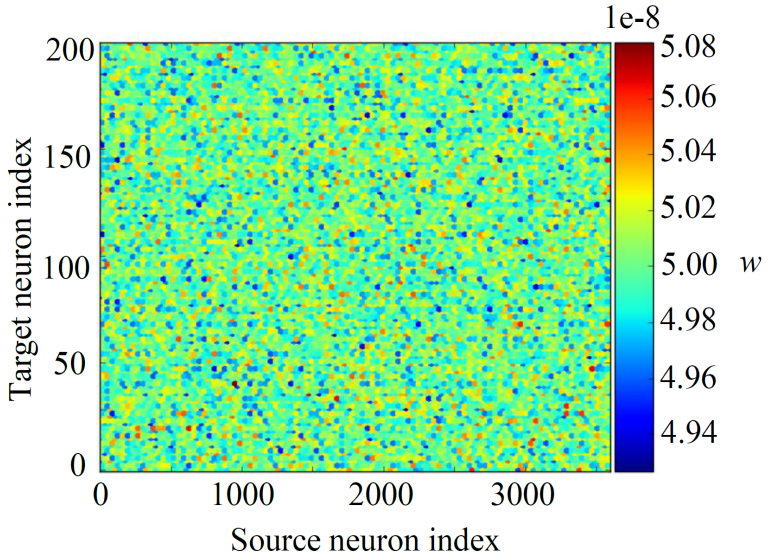


FIGURE 4.30: Weight distribution before training

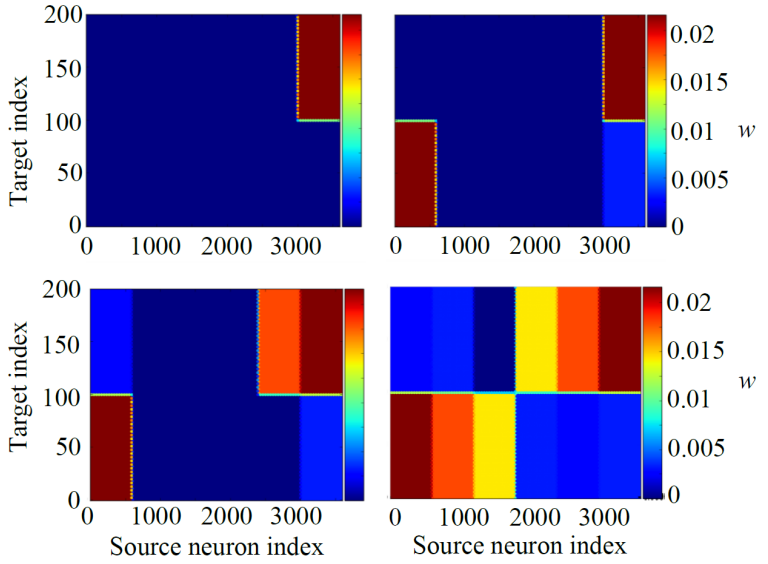


FIGURE 4.31: Weight distribution during training

During the experiment, the figure that the robot receives is shown in Fig. 4.32(a)(c). The image is preprocessed using OpenCV color detection [18], which results in Fig. 4.32(b)(d).

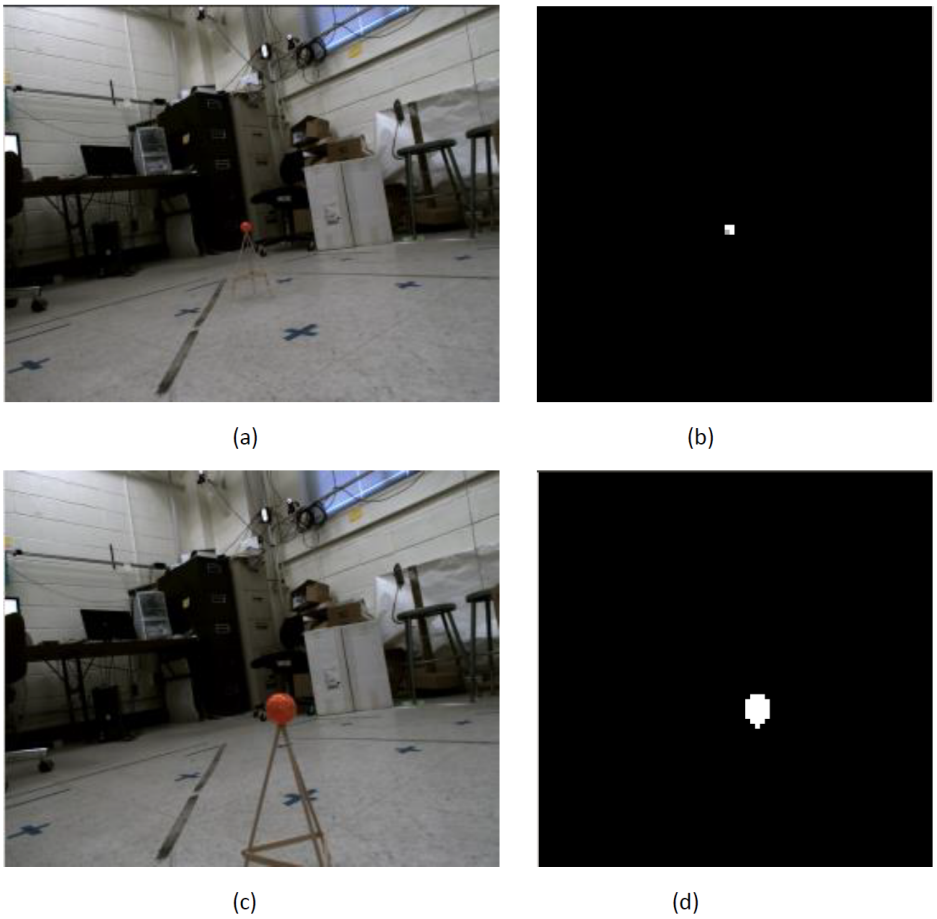


FIGURE 4.32: Camera shots taken from the embedded camera on the robot.(a) Raw image taken from the camera when the target is far away. (b) Preprocessed image using OpenCV when the target is far away. (c) Raw image taken from the camera when the target is nearby. (d) Preprocessed image using OpenCV when the target is nearby.

Fig. 4.32(b)(d) are then sent as an input image to the first layer of the trained SNN. The robot trace during the experiment can be seen in Fig. 4.34. The neural responses during the experiment is shown in Fig. 4.33. This application shows that the indirect training algorithm using teaching signals can indirectly train an large size neural network efficiently for solving target finding task in 3D complex environment.

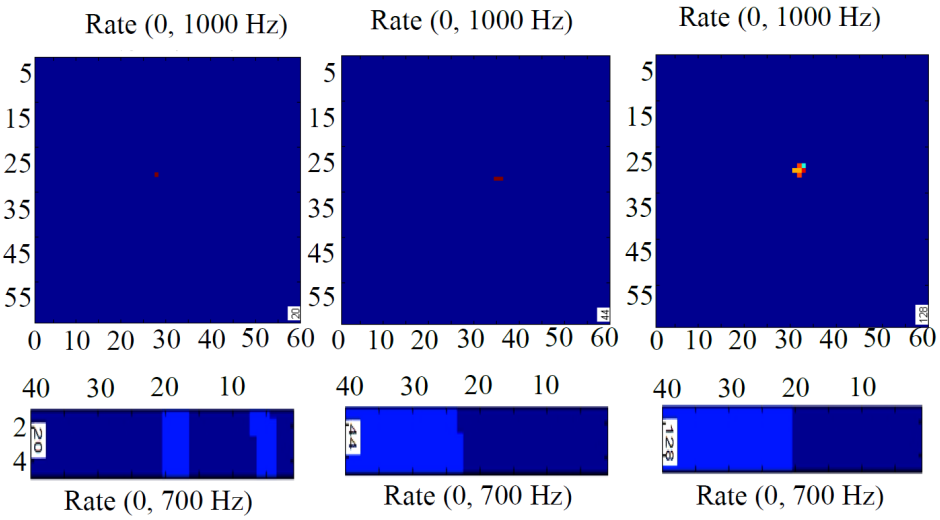


FIGURE 4.33: Snapshots of firings of the large size neural network during robot experiment.

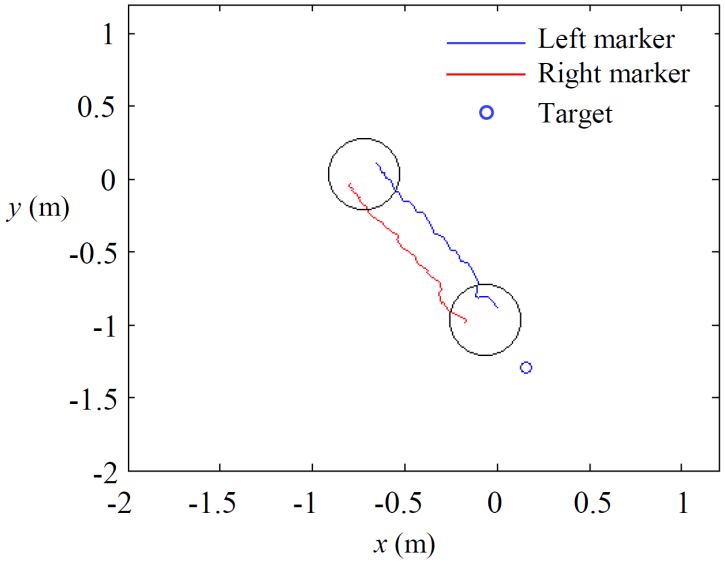


FIGURE 4.34: Snapshots of firings of the large size neural network during robot experiment.

4.3.3 Discussion

While indoor robotic experiments can use *Optitrack* for its localization with millimeter precision [127], the localization of outdoor robots can be done by using GPS and

Radar [20, 2, 122]. However, these equipment can only reconstruct the environment rather than the equivalent of the direct perception of the environment. In addition, his resolution GPS and radar systems can be expensive. Cameras allow for direct perception of the environment are are more cost effective. Understanding the digital images from the camera is a very hard problem that forms a field in computer science called Computer Vision. The motivation of computer vision is to develop algorithms and methods that can be used to understand the environment and automate tasks in a manner consistent with the human visual system. Recent progresses in neural networks such as Alex Net that won 2012 ImageNet competition for image classification, neural network driven cars and amazon robotic competition show a great potential of using neural networks in the field of robotics [88, 25, 29] can be computationally expensive. Therefore, studies on solving real world robot problems like robot navigation, robot interaction and robot perception using spiking neural networks are needed.

Neurorobots can be used to analyze motor feedback and control systems. Locomotion control has been studied using models or central pattern generators, which are a group of neurons for generating repetitive behaviors, to drive four-legged walking robots [76]. However, these are no learning involved in this robot control problem. Some studies examine the memory of rat's hippocampus found that place cells fire at specific locations that has been learned [121, 109]. However, they do not have high density of visual stimuli, which therefore simplified the problem. Recent neurorobot experiment shows a robot with android phone for receiving image inputs and output desired movement can avoid obstacles while reaching an target. However, there is no learning involved in the system. The network's parameters are all set through previous experiment and human experiences [8].

This chapter introduces an indirect training algorithm using supervised teaching signals, which can control the output neurons to fire at certain average firing fre-

quencies during training to let the weights update according to STDP learning rule. In principle, this approach could be applied both before and during navigation. The results show that this algorithm can train a large scale neural networks in a reasonable time. The trained two layer network with thousands of neurons can be used to control a robot with an embedded camera to navigate in real time environment and reach its target. The trained general weights distribution have a decay from the edge of the network to the center of the network to reduce the oscillation of the robot during tests.

For training networks using this method, parameters like A_-, A_+ can affects the performance of the training. When A_-, A_+ are too large the network gives rise an increased speed, which can produce large errors after training. Like artificial neural networks, the network initial weights should be both random and have a small value. The robot's performance also depends on the setting of the OpenCV parameters. A good HSV parameter setting can improve the robot's performance with fewer oscillations in the robot direction.

The present limitation of this algorithm is that only two layer feedforward neural network can be trained. The reason is that the teaching signal can only affect the weights between the stimulated layer and its previous layer given the nature of the stimulus-induced STPD learning in the network. One possible way to improve this is to backpropagate the desired firings from the output layer to the hidden layers so that the algorithm can decide the teaching signals. If this is solved, this algorithm should be able to train a multilayer feedforward neural networks.

5

Indirect ReSuMe Algorithm and Spatial Temporal Mapping

In this chapter I present and analyze the results of temporal and spatial mapping using the indirect ReSuMe algorithm. One of the weaknesses of the standard ReSuMe method is that its STDP-like learning mechanism is not as biological plausible. In the indirect ReSuMe method described here a more realistic version of STDPis used to train a network with 200 input neurons and five output neurons.

5.1 Algorithms Description

Spiking neural networks have been used for two types of problems: rate coding problems and temporal coding problems. Traditional classification, or regression problems such as those studied in the field of deep learning can all be classified as rate coding problem. Temporal coding problem is a special problem that seeks to decode or evaluate the performance of the neural network through measurement of their specific firing times. Temporal coding has been shown to be an important characteristic in human and animals neural system for recognition of color patterns,

visual patterns, odours and sound localization hopfield1995pattern. Studies have shown that the resolution of temporal coding for sound localization is on a millisecond time scale [11, 159]. Visual stimuli are encoded in the latency time between stimulus onset and first spike [63]. Temporal coding is, however, very sensitive to noise [61].

There are several existing methods to train SNNs to solve the spatial and temporal mapping problem based on temporal coding, including SpikeProp [14], Tempotron [69] and the Remote Supervised Method (ReSuMe) [128]. Spikeprop was the first efficient algorithm developed to train a forward spiking neural network to generate output spikes at specific timings. However, the algorithm assumes neurons fire only once during each testing epoch. This assumption limits more complex applications of spiking neural networks. Tempotron enables neurons to learn whether to fire or not to fire for a given set of input stimuli. Therefore, the trained SNNs can do binary classifications. However, the SNN cannot learn specific firing times but instead only learns whether to generate spikes for a given stimulus. ReSuMe is a Hebbian based supervised learning algorithm that uses a STDP-like learning window. Ponulak showed that a random mapping from input spike trains to output spike trains could be achieved. However, the STDP-like rule used in this work was not a biologically plausible for of STDP.

The Indirect ReSuMe algorithm based on a realistic form of STDP is presented below.

5.1.1 Indirect ReSuMe rule

ReSuMe defines a learning algorithm using the following equation:

$$\begin{aligned}\frac{d}{dt}w_{ki}(t) = S^d(t)\left[a^d + \int_0^\infty W^d(s^d)S^{in}(t-s^d)ds^d\right] \\ + S^l(t)\left[a^l + \int_0^\infty W^l(s^l)S^{in}(t-s^l)ds^l\right]\end{aligned} \tag{5.1}$$

where w_{ki} is the synaptic efficacy between input neuron k and output neuron i, a^d, a^l are two constants determining the amplitudes of the weight modifications, S^d, S^l are defined by,

$$S^d(t) = \sum_f \delta(t - t_d^f) \tag{5.2}$$

$$S^l(t) = \sum_f \delta(t - t_l^f) \tag{5.3}$$

where S^d, S^l are the signals represented by spikes, t_d^f is the desired firing time of the output neuron, t_l^f is the actual firing time of output neuron. W^d, W^l in Eqn. 5.1 are two window functions defined by,

$$W^d(s^d) = \begin{cases} +A^d \cdot exp(\frac{-s^d}{\tau_d}) & \text{if } s^d > 0 \\ 0 & \text{if } s^d \leqslant 0 \end{cases} \tag{5.4}$$

$$W^l(s^l) = \begin{cases} -A^l \cdot exp(\frac{-s^l}{\tau_l}) & \text{if } s^l > 0 \\ 0 & \text{if } s^l \leqslant 0 \end{cases} \tag{5.5}$$

where A^d, A^l are positive real values for excitatory synapse and negative real values for inhibitory synapse, τ^d, τ^l are real and positive values, s^d, s^l are defined by,

$$s^d = t_j^{d,f} - t_k^{in,f} \tag{5.6}$$

$$s^l = t_i^{l,f} - t_k^{in,f} \tag{5.7}$$

To simplify the model, the parameters are set: $a^l = -a^d$, $A^l = A^d$ and $\tau^l = \tau^d$, then Eqn. 5.1 becomes,

$$\frac{d}{dt}w_{ki}(t) = [S^d(t) - S^l(t)]\left[a^d + \int_0^\infty W^d(s^d)S^{in}(t - s^d)ds^d\right] \tag{5.8}$$

The equation above shows that the synaptic efficacy is driven by the difference between the desired and generated signals. To implement the indirect form of ReSuME, this synaptic strength changes through STDP rather than through a direct setting of the weight. To accomplish this, multiple pairs of pre- and postsynaptic square pulses are given to the corresponding neurons. The time difference between the square pulses given to neuron i and j is defined as,

$$\delta t = c_i - c_j \tag{5.9}$$

where c_i, c_j are the centers of those square pulses given to neuron i and j, the sign of δ_t is determined by Eqn. 5.8. For example, if $dw_{ki} > 0$, the δ_t will be set negative so that the presynaptic neuron fires before the postsynaptic neuron in order to increase the weight and visa versa. Small A_+, A_- in STDP rule are chosen so that dw_{ki} for all k, i can be indirectly implemented through STDP with a good precision.

$$\delta e_w = \delta w - \delta w^* < E_w \tag{5.10}$$

where δe_w is the error difference between the actual weight change through indirect training δw and the desired weight change through ReSuMe δw^*, $E_w = 1e - 10$ is the criteria value. The pseudo code Alg. 3 can be found in Appendix. A.

5.2 Results of Training Biological Realistic Model for Temporal Spatial Mapping

The network is a two layer feedforward network. The input layer has 200 neurons while the output layer has 5 neurons. The connections are all-to-all style.

The neuron model used in solving this problem is Izhikevich neuron model and STDP learning rule. The synaptic weights during training is shown in Fig. 5.5. The initial weights are set to be 80 % positive (excitatory) and 20 % negative (inhibitory).

Fig. 5.1 shows the actual spikes of the output neurons and their desired spike timings before training. Fig. 5.2 shows the actual spikes and the desired spikes after training using ReSuMe method, which directly set the synaptic weights. The results show that the training can let the output neurons fire at desired spike timings.

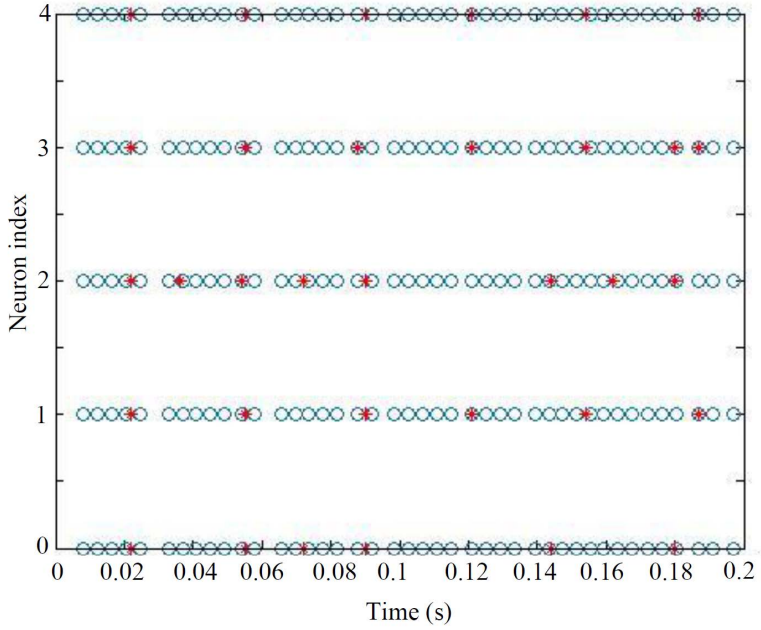


FIGURE 5.1: Raster plot of output neurons before training using ReSuMe algorithm given random generated input spikes.

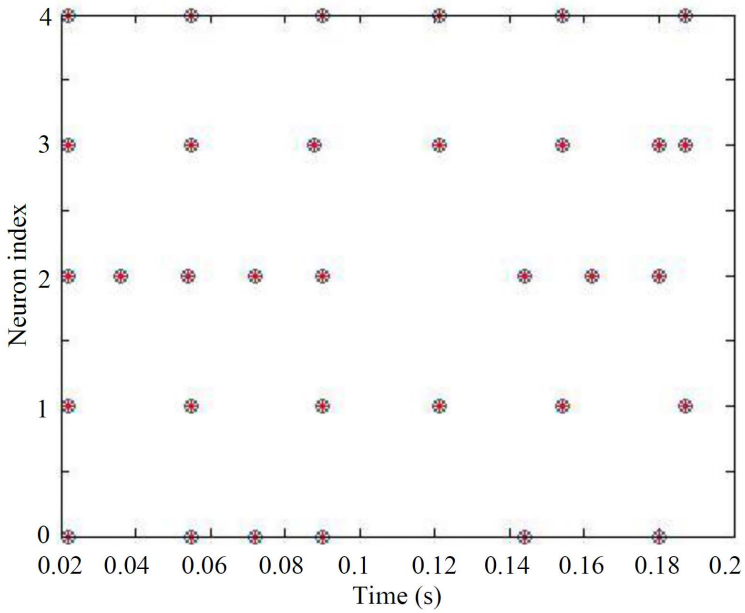


FIGURE 5.2: Raster plot of output neurons after training using ReSuMe algorithm given random generated input spikes.

Fig. 5.3 shows the actual spikes and the desired spikes before training using the indirect ReSuMe. Fig. 5.4 shows the actual spikes and the desired spikes after training using the indirect ReSuMe method. The accuracy improves comparing with the firings before training even though it's lower than the ReSuMe method. The weight updates according to the STDP during the indirect training are plotted in Fig. 5.5.

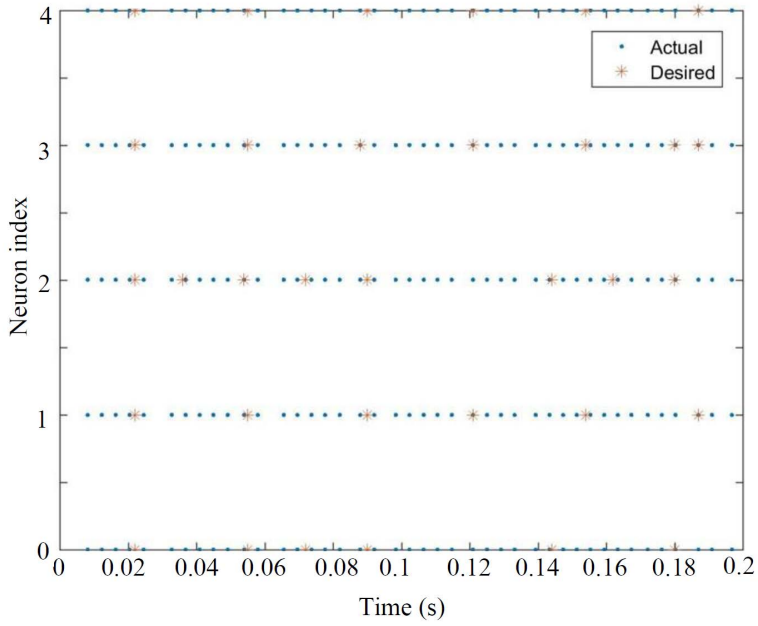


FIGURE 5.3: Raster plot of output neurons before training given random generated input spikes.

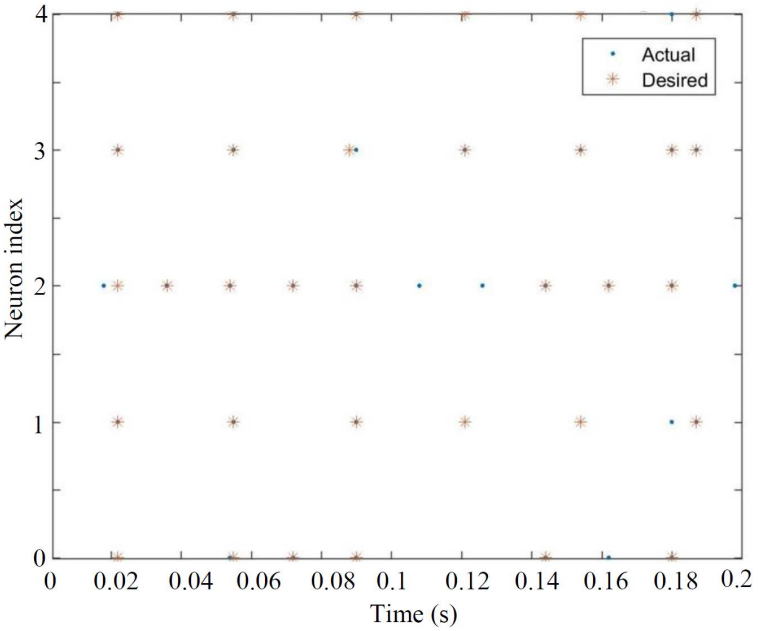


FIGURE 5.4: Raster plot of output neurons after training given random generated input spikes.

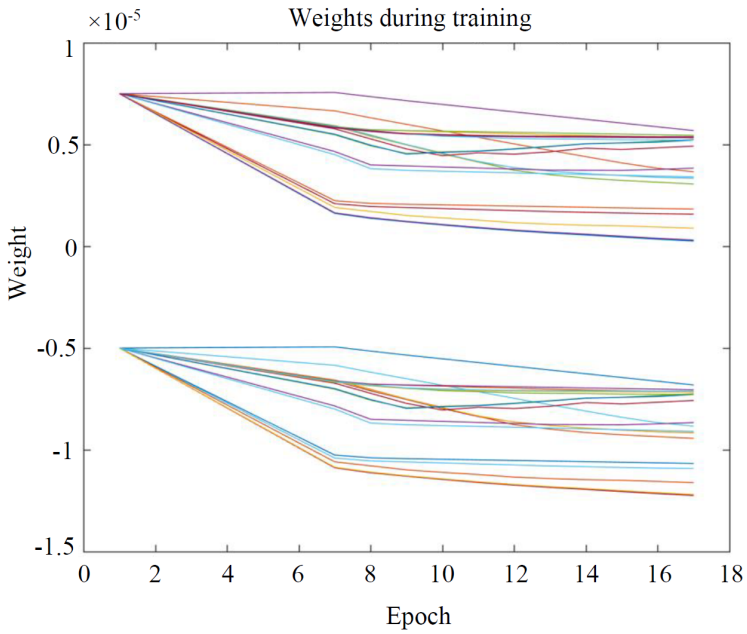


FIGURE 5.5: The weight change during indirect training for spatial and temporal mapping.

Fig. 5.6 shows the performance measured by Correlation-based metric defined in Chapter 4 during training using ReSuMe method. All the output neuron's performances improve during training.

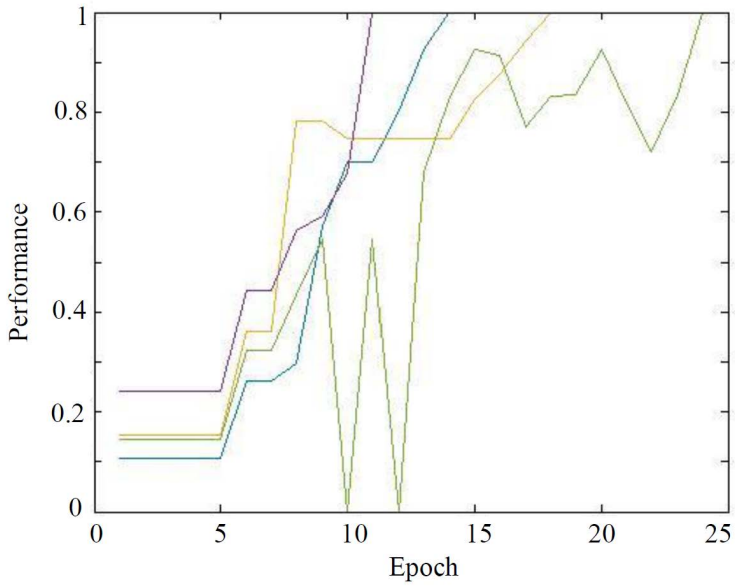


FIGURE 5.6: The performance of five output neurons during training using ReSuMe.

Fig. 5.7 shows the performance measured by Correlation-based metric defined in Chapter 4 during training using indirect ReSuMe method. All the output neurons performances improve during training.

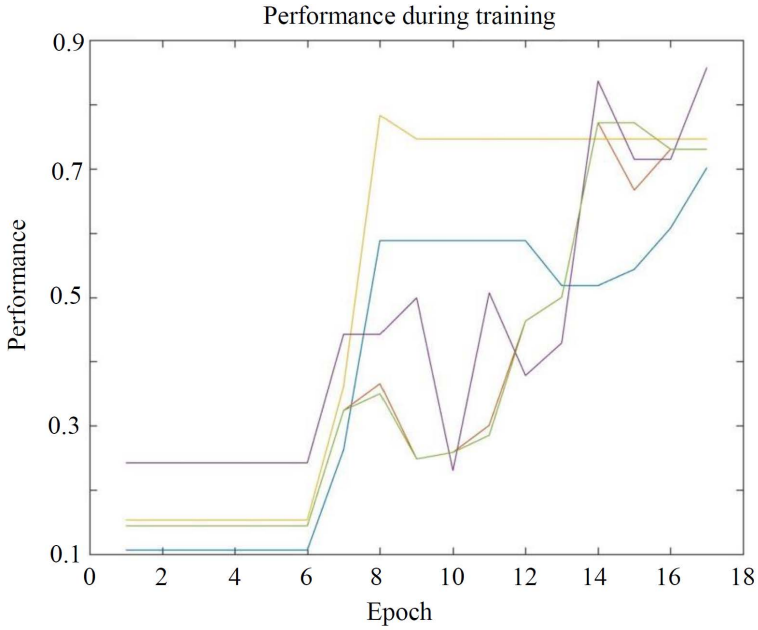


FIGURE 5.7: The performance of five output neurons during training.

5.2.1 *Analysis of the Indirect Learning Rate*

In order to find what variable affects the performance of indirect ReSuMe, multiple simulations are run with different values of the A_+ parameter in the STDP model. Fig. 5.8 shows that the performance decrease as the A_+ increases. The reason is that the pair based firings of the pre and postsynaptic neurons cannot drive weights to their desired weights if the A_+ is too large.

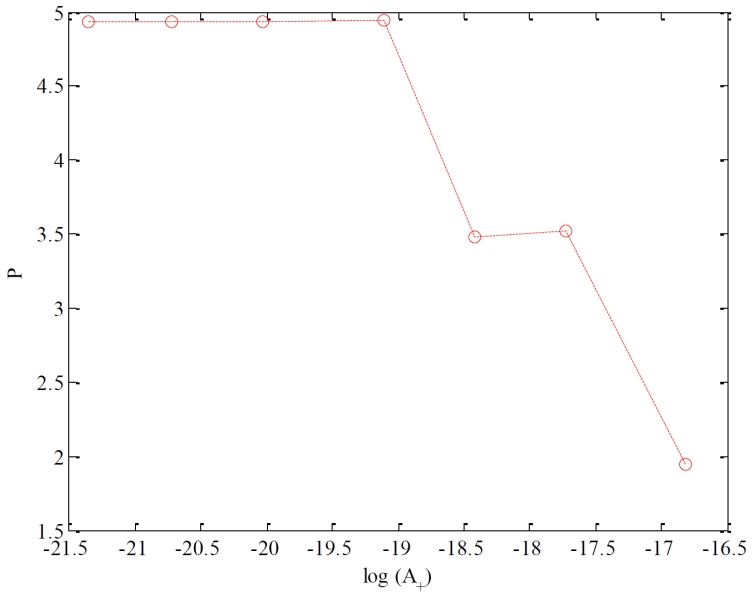


FIGURE 5.8: Performance analysis of indirect ReSuMe under different STDP parameters.

The decrease of A_+ can improve the final performance after training using indirect ReSuMe. But this can increase the time it takes to train the network due to the number of square pulses increases. Fig. 5.9 shows the time length it takes to train the SNN given different values of A_+.

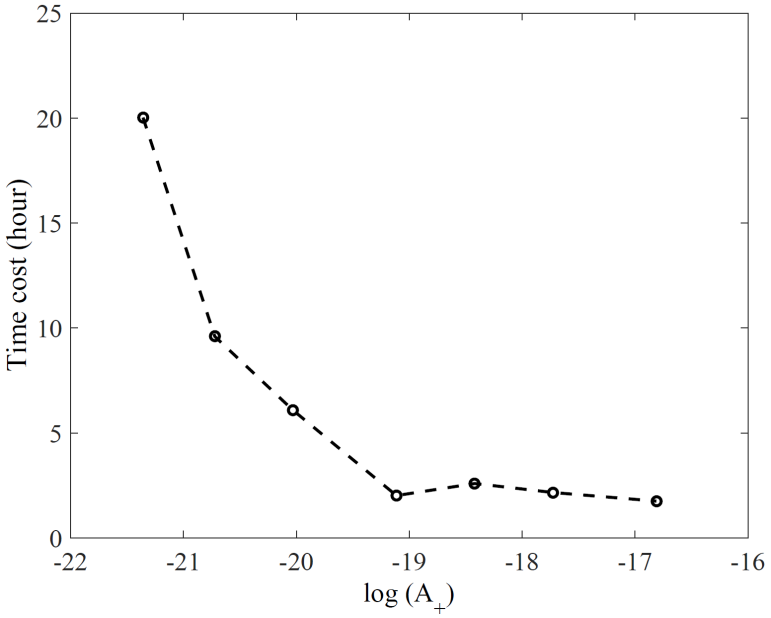


FIGURE 5.9: Time cost analysis of indirect ReSuMe under different STDP parameters.

Finally, the analysis of the effects of network size on the final performance is plotted in Fig. . It can be seen that the training algorithm has a good scalability.

5.3 Discussion

This chapter describes an indirect training algorithm to train an spiking neural network based on a biologically realistic STDP learning rule for spatial and temporal mapping of output neuron's spikes. The algorithm is derived from standard ReSuMe algorithm, which uses a non-biologically plausible version of STDP. As with the perturbation methods, the indirect ReSuMe method uses paired stimulation with square pulses to modify the weights. The indirect ReSuMe does not allow the synapses to change from inhibitory to excitatory or vice versa during training, which is also more biologically plausible.

The ReSuMe algorithm can train the network with good performance. The performance of the indirect ReSuMe algorithm was found to improved with smaller

parameters for A_+, A_-. However, there is a trade off between the performance of the indirect ReSuMe and the time needed for training. When small values of A_+, A_- are used, the performance can be almost as precise as the ReSuMe algorithm. Unfortunately, it can takes days to train the network. Temporal coding is found in many parts of the biological neural systems. Because of being able to solve the spatial and temporal mapping based on temporal coding problem, indirect ReSuMe is needed to be applied to those networks. Its potential application could be neuroprosthetics, neurorobotics and neuromorphic chips. Indirect ReSuMe can also be applied to train recurrent neural networks by training only synapses that connect the hidden pool and the output pool.

As noted, the main limitation of the current indirect ReSuMe method is the time for training. One way to reduce the training time is to stimulate neurons using square pulses that have various temporal differences rather than only a sign difference. This should be able to largely reduce the time cost due to multiple pairs of square pulses needed during each training epoch. This improvement can also improve the performance of the indirect ReSuMe because it can reduce the error between the desired weight change and the actual induced weight change.

6

Conclusions

Four different indirect training methods are presented, which induce changes in synaptic plasticity by controlled pulses abiding by the STDP rule, as opposed to direct weight manipulation. The idea of training neural networks by controlling cell activity is motivated by emerging techniques in neuroscience, such as optogenetics, with enable precise spatio-temporal control of cell activity using light. The indirect training algorithms use patterns of square pulse stimuli to drive plasticity and can be optimized to optimized to generate the firing rates or even precise spike timings that minimize a desired objective function. The square pulse stimuli can cause a single neuron to fire precisely at desired time with millisecond precision. This property enables the precise firing of pairs of neurons so that the strength of any possible synapse between them can be increased or decreased through spike time dependent plasticity. SNNs trained this way can be used to control both virtual and real world robots to navigate in an unknown terrain with obstacles.

In this thesis, SNNs using simplified and biologically realistic models of neurons and synapses are used to test the different training algorithms. The Izhikevich model can reproduce spiking and bursting behavior of multiple types of cortical neurons.

It can replicate Hodgkin-Huxley type dynamics with low computational complexity similar to integrate-and-fire model. Conductance based synapse model is more biologically plausible than simple Dirac pulse. The application of the indirect methods to more biological realistic models suggests that these algorithms to be used on cultures or real neurons.

The indirect SGD algorithm improves the performance by relying on the gradient rather than only the directions of the weight update. Compared to indirect perturbation, indirect SCG shows better performance when training the same size networks to controlling the virtual insect with large sensor noise. To train SNNs for spatial and temporal mapping of spikes, an indirect ReSuMe algorithm is developed by incorporating a more realistic form of STDP learning rule in the standard ReSuMe algorithm by stimulating the neurons with square pulses so that the output neurons can fire at desired timings for a randomly generated set of input spikes. Even though the performance of the indirect method is not as good as the ReSuMe method, it's shown that the performance can be improved by using smaller STDP parameters. Finally, the indirect training by supervised teaching signals method can train the largest SNN (with almost 4000 neurons) known to date. Teaching signals are given to a region of neurons other than a single neuron to control those neurons to fire in a certain frequency, which can drive the desired synapses to their desired weights. This algorithm is applied to controlling a neurorobot with an embedded camera to find its desired colored target. The indirect SGD and indirect perturbation algorithm both have potential application on training networks of caged neuron non chips if neurons that are not stimulated during training can be silent or inhibited. One advantage of the indirect SGD method over other methods is that it can train both feedforward multilayer neural network and recurrent neural networks. While the training time can be significant on large network, it can be shortened using faster processors such as GPU accelerated implementation of SNNs like that used in Carlsim.

The indirect stochastic gradient descent algorithm and indirect training through teaching signals are two most promising algorithms to be applied to larger networks and larger data sets. Compared with indirect weight perturbation algorithm, indirect SGD algorithm can converge to lower error given the same network structure. The Indirect training through teaching signals method was shown to train the largest feedforward network efficiently. The efficiency of the other indirect algorithms is limited because they use the square pulse stimuli to control the firing time of pairs of single neurons very precisely. For future study, the square pulse's center can be adjusted by specific timings rather than flipping of the sign, which might largely reduce the time to train SNNs using indirect SGD or indirect ReSuMe.

The simulation of virtual insect path planning shows that these algorithms can train SNN to approximate the mapping between the input and desired output, which enables the SNN to solve control problems like path planning both in biological neuronal networks, and in CMOS/memristor nanoscale chip. These algorithms have been validated in hardware, which were accomplished both at the FPGA level and the CMOS level [74, 111]. The implemented design was tested on real hardware to show that the proposed SNN structure and training algorithm can be adopted in circuit designs. These algorithms have also been tested on neurorobotic navigation problems using both Optitrack and embedded camera with analysis of different level of sensor noise.

Future work may include application of indirect training algorithms on larger size networks for solving more complex problems of image recognition. Because the networks needed to accomplish certain tasks may need to be large. The use of a momentum term in the indirect SGD algorithm may reduce the time it takes to train large networks with large input datasets. At present the methods have only be tested in relatively simple network topologies. The modeling of the neural system for controlling the neurorobots could be designed with an architecture to mimic that

of an insect which operate in a dynamic changing environment.

One of the most successful ANNs, convolutional neural networks (CNNs) mimic the multilayer processing in the visual system. Some papers have trained multilayer SNNs with same structure as CNNs by mapping the trained CNNs weights direct to the SNNs. The trained SNNs can classify MNIST dataset in a high accuracy comparing with all other algorithms. However, this cannot be considered as a SNN training because it did not use any biologically plausible learning rules like STDP and Hebbian learning. Therefore our indirect training by supervised teaching signal algorithm can be used to train an SNN with convolutional structure for classifying large image datasets.

Further neurorobotic experiments can use the indirect training algorithm though teaching signals by combining with multilayer trained CNNs as a replacement of OpenCV color detection. This can not only reduce the effects of noise from other objects with similar colors and also solve more complex tasks by recognizing complex shapes in the environment. After improving the indirect training by supervised teaching signal algorithm by back-propagating the desired firing frequency, the indirect training algorithm should be able to train deep multilayer SNNs efficiently.

Finally, none of the SNNs have been applied to solve the autonomous driving problem. Even though some companies, like Tesla, have invented autonomous driving based on using Convolutional Neural Networks, they are currently limited and such techniques are not ready to completely replace human drivers. In addition, CNNs require implementation of GPUs. SNNs can be implemented in neuromorphic chips and can trained using the indirect training algorithms developed here. We have already presented preliminary evidence that our indirect training algorithms can be applied on CMOS/memristor based chips, which may form the basis of the next generation of intelligent systems.

Appendix A

Appendix

Table A.1: Parameters for neuron models for Chapter 4 and Chapter 5.

Parameter	Value during simulation
$C_m(F)$	3e-8
$R_m(Ohm)$	1e6
$V_{thresh}(V)$	0.017
$V_{resting}(V)$	0.014
$V_{init}(V)$	0
$T_{refract}(s)$	0.002
$I_{noise}(A)$	5e-12

Table A.2: Parameters for neuron models for Chapter 6.

Parameter	Value during simulation
τ_m	10 ms
$R_m(Ohm)$	1e7
$V_{thresh}(V)$	-55
$V_{resting}(V)$	-65
$V_{init}(V)$	0
$T_{refract}(s)$	0.004
$I_{noise}(A)$	0

The units for the variables are shown in Table A.4.

Table A.3: Parameters for neuron models for Chapter 7.

Parameter	Value during simulation
$\tau_+(ms)$	20
$\tau_-(ms)$	60
A_+	0.1
A_-	-0.03
a	0.02
b	0.2
c	-65
d	8

Table A.4: Units for parameters

Parameter	Unit
$t, t_L, t_R, T_{refract}, t_r, \beta, c_{i,l}, b_0, b_k$	s
$x, y, L, d_L, d_R, \sigma, D(a,b), \lambda$	mm
α	A/mm
$h_L, g_L, h_R, g_R, s(t), S_i, S_i^0, I_{noise}, \omega$	A
v, v_L, v_R, η	mm/s
$f, f_L^*, f_R^*, f_L, f_R, z_i, f_L^0, f_R^0$	Hz
C_m	F
R_m	ohm
$V_{thresh}, V_{resting}, V_{init}$	V

Result: A trained SNN
Create an SNN;
while $e_{k,2}$>*criteria* **do**
    Test the SNN;
    calculate $e_{k,1}$;
    Pick neuron i, j;
    Generate s_i, s_j with temporal difference b_0;
    Stimulate i and j;
    Test the SNN;
    calculate $e_{k,2}$;
    if $e_{k,2}$>$e_{k,1}$ **then**
        Swap s_i, s_j;
    else
        Keep s_i, s_j;
    end
    Stimulate i and j;
    k=k+1;
end

Algorithm 1: Indirect perturbation algorithm pseudo code.

Data: Initialize the training cases
Result: A trained SNN
initialization;
Randomize pairs into subsets W_{batch};
$batch = 0$;
while $E > criteria$ **do**
  Run six different cases;
  Calculate E;
  if $batch > 100$ **then**
    randomize pairs into subsets W_{batch}
  end
  for i *and* j, $i, j \in W_{batch}$ **do**
    Stimulate i and j with time difference δt;
    Run six different cases;
    Calculate $E^{ij}_{perturb}$;
    Calculate gradient $G_{ij} = \frac{E^{ij}_{perturb} - E}{A^+ \exp \frac{\delta t}{\tau_+}}$
  end
  for i *and* j, $i, j \in W_{batch}$ **do**
    Calculate $\delta w^*_{ij} = -\eta * G_{ij}$;
    Calculate number of spike pairs needed N_{spair};
    for $i = 1 : N_{spair}$ **do**
      Stimulate i and j;
    end
  end
  $batch+ = 1$;
end

Algorithm 2: Indirect stochastic gradient descent algorithm pseudo code.

Initialize W and desired output timings;
Initialize input I **while** $min(C) > criteria$ **do**
    Run a test for 200 ms;
    for j *in output neuron index* **do**
        Calculate correlation $C[j] = \frac{v_d \cdot v_0}{|v_d||v_0|}$;
        Calculate $\frac{d}{dt} w_{ki}(t)$ using Eq. 5.1;
        while $max(abs(e_w)) > E_w$ **do**
            for $i = 1 : 200$ **do**
                Calculate desired $delta_t = sign(e_w[i])$;
                Calculate desired c_i, c_j;
                Stimulate each i, j with square pulses centered c_i, c_j;
                Update $e_w[i] = \delta w - \delta w^*$;
            end
        end
    end
    $batch+ = 1$;
end

Algorithm 3: Indirect ReSuMe algorithm pseudo code.

```
Initialize the network W; for i = 1 : 6 do
    Initialize input current I_i for input region i in input layer;
    Initialize desired output f_i^* for case i;
    e=100;
    while abs(e)>criteria do
        Turn off STDP;
        initialize S_k = 3;
        while f_t^* − f_t >criteria2 do
            Stimulate input region i and output region using I_i and S_k;
            if abs(f_t^* − f_t) <= criteria2 then
                break;;
            end
            if abs(f_t^* − f_t) > 0 then
                S_k = S_{k−1} + α ∗ abs(f_t^* − f_t);
            else
                S_k = S_{k−1} − α ∗ abs(f_t^* − f_t);
            end
            k=k+1;
        end
        Turn on STDP;
        Stimulate input neurons and output regions using I_i and S_k;
        Update e = f_i^* − f_i
    end
end
```

Algorithm 4: Indirect ReSuMe algorithm pseudo code.

Bibliography

[1] Abbott, L., and Kepler, T. B. Model neurons: From hodgkin-huxley to hopfield. In *Statistical mechanics of neural networks*. Springer, 1990, pp. 5–18.

[2] Agrawal, M., and Konolige, K. Real-time localization in outdoor environments using stereo vision and inexpensive gps. In *18th International Conference on Pattern Recognition (ICPR'06)* (2006), vol. 3, IEEE, pp. 1063–1068.

[3] Azevedo, F. A., Carvalho, L. R., Grinberg, L. T., Farfel, J. M., Ferretti, R. E., Leite, R. E., Lent, R., Herculano-Houzel, S., et al. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *Journal of Comparative Neurology 513*, 5 (2009), 532–541.

[4] Bajpai, P., and Kumar, M. Genetic algorithm–an approach to solve global optimization problems.

[5] Bakkum, D. J., Gamblen, P. M., Ben-Ary, G., Chao, Z. C., and Potter, S. M. Meart: the semi-living artist. *Frontiers in neurorobotics 1* (2007), 5.

[6] Belatreche, A., Maguire, L., McGinnity, M., and Wu, Q. A method for supervised training of spiking neural networks. In *Proc. IEEE Conf. Cybernetics Intelligence–Challenges and Advances, CICA* (2003), Citeseer, pp. 39–44.

[7] Beyeler, M., Dutt, N. D., and Krichmar, J. L. Categorization and decision-making in a neurobiologically plausible spiking network using a stdp-like learning rule. *Neural Networks 48* (2013), 109–124.

[8] Beyeler, M., Oros, N., Dutt, N., and Krichmar, J. L. A gpu-accelerated cortical neural network model for visually guided robot navigation. *Neural Networks 72* (2015), 75–87.

[9] Bi, G.-q., and Poo, M.-m. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *The Journal of neuroscience 18*, 24 (1998), 10464–10472.

[10] Bi, G.-q., and Poo, M.-m. Synaptic modification by correlated activity: Hebb's postulate revisited. *Annual review of neuroscience 24*, 1 (2001), 139–166.

[11] Blauert, J. *Spatial hearing: the psychophysics of human sound localization*. MIT press, 1997.

[12] Bliss, T. V., and Lømo, T. Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path. *The Journal of physiology 232*, 2 (1973), 331–356.

[13] Boahen, K. Neuromorphic microchips. *Scientific American 292*, 5 (2005), 56–63.

[14] Bohte, S. M., Kok, J. N., and La Poutre, H. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing 48*, 1 (2002), 17–37.

[15] Booij, O., and tat Nguyen, H. A gradient descent rule for spiking neurons emitting multiple spikes. *Information Processing Letters 95*, 6 (2005), 552–558.

[16] Boyden, E. S., Zhang, F., Bamberg, E., Nagel, G., and Deisseroth, K. Millisecond-timescale, genetically targeted optical control of neural activity. *Nature neuroscience 8*, 9 (2005), 1263–1268.

[17] Brader, J. M., Senn, W., and Fusi, S. Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural computation 19*, 11 (2007), 2881–2912.

[18] Bradski, G., et al. The opencv library.

[19] Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., Diesmann, M., Morrison, A., Goodman, P. H., Harris Jr, F. C., et al. Simulation of networks of spiking neurons: a review of tools and strategies. *Journal of computational neuroscience 23*, 3 (2007), 349–398.

[20] Bulusu, N., Heidemann, J., and Estrin, D. Gps-less low-cost outdoor localization for very small devices. *IEEE personal communications 7*, 5 (2000), 28–34.

[21] Buonomano, D. V., and Maass, W. State-dependent computations: spatiotemporal processing in cortical networks. *Nature Reviews Neuroscience 10*, 2 (2009), 113–125.

[22] Buračas, G. T., and Albright, T. D. Gauging sensory representations in the brain. *Trends in neurosciences 22*, 7 (1999), 303–309.

[23] Caporale, N., and Dan, Y. Spike timing-dependent plasticity: a hebbian learning rule. *Annual Review of Neuroscience 31* (2008), 25–46.

[24] Carpenter, G. A., and Grossberg, S. *Pattern recognition by self-organizing neural networks*. MIT Press, 1991.

[25] Chen, C., Seff, A., Kornhauser, A., and Xiao, J. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision* (2015), pp. 2722–2730.

[26] Cireşan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. High-performance neural networks for visual object classification. *arXiv preprint arXiv:1102.0183* (2011).

[27] Connors, B. W., and Gutnick, M. J. Intrinsic firing patterns of diverse neocortical neurons. *Trends in neurosciences 13*, 3 (1990), 99–104.

[28] Corneil, D., Sonnleithner, D., Neftci, E., Chicca, E., Cook, M., Indiveri, G., and Douglas, R. Function approximation with uncertainty propagation in a vlsi spiking neural network. In *Neural Networks (IJCNN), The 2012 International Joint Conference on* (2012), IEEE, pp. 1–7.

[29] Correll, N., Bekris, K. E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J. M., and Wurman, P. R. Lessons from the amazon picking challenge. *arXiv preprint arXiv:1601.05484* (2016).

[30] Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems 2*, 4 (1989), 303–314.

[31] Dan, Y., and Poo, M.-m. Hebbian depression of isolated neuromuscular synapses in vitro. *Science 256*, 5063 (1992), 1570–1573.

[32] DasGupta, B., and Schnitger, G. The power of approximating: a comparison of activation functions. *MATHEMATICAL RESEARCH 79* (1994), 641–641.

[33] Dayan, P., and Abbott, L. Theoretical neuroscience: computational and mathematical modeling of neural systems. *Journal of Cognitive Neuroscience 15*, 1 (2003), 154–155.

[34] Deisseroth, K. Optogenetics. *Nature Methods 8* (2011), 26–29.

[35] DeMarse, T. B., Wagenaar, D. A., Blau, A. W., and Potter, S. M. The neurally controlled animat: biological brains acting with simulated bodies. *Autonomous robots 11*, 3 (2001), 305–310.

[36] Destexhe, A., Babloyantz, A., and Sejnowski, T. J. Ionic mechanisms for intrinsic slow oscillations in thalamic relay neurons. *Biophysical Journal 65*, 4 (1993), 1538.

[37] Diehl, P. U., and Cook, M. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience 9* (2015).

[38] Douglas, R. M., and Goddard, G. V. Long-term potentiation of the perforant path-granule cell synapse in the rat hippocampus. *Brain research 86*, 2 (1975), 205–215.

[39] Ebong, I. E., and Mazumder, P. Cmos and memristor-based neural network design for position detection. *Proceedings of the IEEE 100*, 6 (2012), 2050–2060.

[40] El-Husseini, A. E.-D., Schnell, E., Dakoji, S., Sweeney, N., Zhou, Q., Prange, O., Gauthier-Campbell, C., Aguilera-Moreno, A., Nicoll, R. A., and Bredt, D. S. Synaptic strength regulated by palmitate cycling on psd-95. *Cell 108*, 6 (2002), 849–863.

[41] Endo, G., Morimoto, J., Matsubara, T., Nakanishi, J., and Cheng, G. Learning cpg-based biped locomotion with a policy gradient method: Application to a humanoid robot. *The International Journal of Robotics Research 27*, 2 (2008), 213–228.

[42] Erickson, J., Tooker, A., Tai, Y.-C., and Pine, J. Caged neuron mea: A system for long-term investigation of cultured neural network connectivity. *Journal of neuroscience methods 175*, 1 (2008), 1–16.

[43] Ferrari, S., Mehta, B., Muro, G. D., VanDongen, A. M., and Henriquez, C. Biologically realizable reward-modulated hebbian training for spiking neural networks. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on* (2008), IEEE, pp. 1780–1786.

[44] Ferrari, S., Mehta, B., Muro, G. D., VanDongen, A. M., and Henriquez, C. Biologically realizable reward-modulated hebbian training for spiking neural networks. *Proc. International Joint Conference on Neural Networks, Hong Kong* (2008), 1781–1787.

[45] Ferster, D., and Spruston, N. Cracking the neuronal code. *Science 270*, 5237 (1995), 756.

[46] Fields, R. D. Myelination: an overlooked mechanism of synaptic plasticity? *The Neuroscientist 11*, 6 (2005), 528–531.

[47] Fiete, I. R., and Seung, H. S. Gradient learning in spiking neural networks by dynamic perturbation of conductances. *Physical review letters 97*, 4 (2006), 048104.

[48] FitzHugh, R. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical journal 1*, 6 (1961), 445.

[49] Fletcher, T. L., Cameron, P., De Camilli, P., and Banker, G. The distribution of synapsin i and synaptophysin in hippocampal neurons developing in culture. *The Journal of neuroscience 11*, 6 (1991), 1617–1626.

[50] Florian, R. V. Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation 19*, 6 (2007), 1468–1502.

[51] Florian, R. V. The chronotron: a neuron that learns to fire temporally precise spike patterns. *PloS one 7*, 8 (2012), e40233.

[52] Foderaro, G., Henriquez, C., and Ferrari, S. Indirect training of a spiking neural network for flight control via spike-timing-dependent synaptic plasticity. In *Decision and Control (CDC), 2010 49th IEEE Conference on* (2010), IEEE, pp. 911–917.

[53] Friedrich, J., Urbanczik, R., and Senn, W. Code-specific learning rules improve action selection by populations of spiking neurons. *International journal of neural systems 24*, 05 (2014), 1450002.

[54] Funahashi, K.-i., and Nakamura, Y. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks 6*, 6 (1993), 801–806.

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

[56] Gerstner, W., Kreiter, A. K., Markram, H., and Herz, A. V. Neural codes: firing rates and beyond. *Proceedings of the National Academy of Sciences 94*, 24 (1997), 12740–12741.

[57] Ghosh-Dastidar, S., and Adeli, H. Improved spiking neural networks for eeg classification and epilepsy and seizure detection. *Integrated Computer-Aided Engineering 14*, 3 (2007), 187–212.

[58] Ghosh-Dastidar, S., and Adeli, H. A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. *Neural Networks 22*, 10 (2009), 1419–1431.

[59] Ghosh-Dastidar, S., and Adeli, H. Spiking neural networks. *International journal of neural systems 19*, 04 (2009), 295–308.

[60] Gibson, J. R., Beierlein, M., and Connors, B. W. Two networks of electrically coupled inhibitory neurons in neocortex. *Nature 402*, 6757 (1999), 75–79.

[61] Gilles, W., Michèle, T., and Khashayar, P. Intrinsic variability of latency to first-spike. *Biological cybernetics 103*, 1 (2010), 43–56.

[62] Goldberg, D. Genetic algorithms in search, optimization, and machine learning, 1989.

[63] Gollisch, T., and Meister, M. Rapid neural coding in the retina with relative spike latencies. *science 319*, 5866 (2008), 1108–1111.

[64] Goodman, D., and Brette, R. Brian: a simulator for spiking neural networks in python.

[65] Gori, M., and Tesi, A. On the problem of local minima in backpropagation. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 1 (1992), 76–86.

[66] Graf, H. P., and Henderson, D. A reconfigurable cmos neural network. In *Solid-State Circuits Conference, 1990. Digest of Technical Papers. 37th ISSCC., 1990 IEEE International* (1990), IEEE, pp. 144–145.

[67] Gray, C. M., and McCormick, D. A. Chattering cells: superficial pyramidal neurons contributing to the generation of synchronous oscillations in the visual cortex. *Science 274*, 5284 (1996), 109–113.

[68] Grüning, A., and Bohte, S. M. Spiking neural networks: Principles and challenges. In *ESANN* (2014).

[69] Gütig, R., and Sompolinsky, H. The tempotron: a neuron that learns spike timing–based decisions. *Nature neuroscience 9*, 3 (2006), 420–428.

[70] Hebb, D. O. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[71] Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural networks 2*, 5 (1989), 359–366.

[72] Hsu, J. Ibm's new brain [news]. *Spectrum, IEEE 51*, 10 (2014), 17–19.

[73] Hu, D., Zhang, X., Xu, Z., Ferrari, S., and Mazumder, P. Digital implementation of a spiking neural network (snn) capable of spike-timing-dependent plasticity (stdp) learning. In *Nanotechnology (IEEE-NANO), 14th IEEE International Conference on* (2014), IEEE , pp. 873–876.

[74] Hu, D., Zhang, X., Xu, Z., Ferrari, S., and Mazumder, P. Digital implementation of a spiking neural network (snn) capable of spike-timing-dependent plasticity (stdp) learning. In *14th IEEE International Conference on Nanotechnology* (2014), IEEE, pp. 873–876.

[75] Iannella, N., and Back, A. D. A spiking neural network architecture for nonlinear function approximation. *Neural networks 14*, 6 (2001), 933–939.

[76] Ijspeert, A. J., Crespi, A., Ryczko, D., and Cabelguen, J.-M. From swimming to walking with a salamander robot driven by a spinal cord model. *science 315*, 5817 (2007), 1416–1420.

[77] Inagaki, S., Yuasa, H., Suzuki, T., and Arai, T. Wave cpg model for autonomous decentralized multi-legged robot: Gait generation and walking speed control. *Robotics and Autonomous Systems 54*, 2 (2006), 118–126.

[78] Izhikevich, E. M. Solving the distal reward problem through linkage of stdp and dopamine signaling. In *Cerebral cortex* (2007).

[79] Izhikevich, E. M., et al. Simple model of spiking neurons. *IEEE Transactions on neural networks 14*, 6 (2003), 1569–1572.

[80] Izhikevich, E. M., and Moehlis, J. Dynamical systems in neuroscience: The geometry of excitability and bursting. *SIAM review 50*, 2 (2008), 397.

[81] Joshi, P., and Triesch, J. Optimizing generic neural microcircuits through reward modulated stdp. In *International Conference on Artificial Neural Networks* (2009), Springer, pp. 239–248.

[82] Kasabov, N., Dhoble, K., Nuntalid, N., and Indiveri, G. Dynamic evolving spiking neural networks for on-line spatio-and spectro-temporal pattern recognition. *Neural Networks 41* (2013), 188–201.

[83] Kepler, T. B., Abbott, L., and Marder, E. Reduction of conductance-based neuron models. *Biological cybernetics 66*, 5 (1992), 381–387.

[84] Kimura, H., Fukuoka, Y., and Cohen, A. H. Biologically inspired adaptive walking of a quadruped robot. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 365*, 1850 (2007), 153–170.

[85] Koickal, T. J., Hamilton, A., Pearce, T. C., Tan, S. L., Covington, J. A., and Gardner, J. W. Analog vlsi design of an adaptive neuromorphic chip for olfactory systems. In *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on* (2006), IEEE, pp. 4–pp.

[86] Kopell, N., Ermentrout, G., and Williams, T. On chains of oscillators forced at one end. *SIAM Journal on Applied Mathematics 51*, 5 (1991), 1397–1417.

[87] Krichmar, J. Neurorobotics. *Scholarpedia 3*, 3 (2008), 1365. revision 152121.

[88] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[89] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[90] Legenstein, R., Chase, S. M., Schwartz, A. B., and Maass, W. A reward-modulated hebbian learning rule can explain experimentally observed network reorganization in a brain control task. *The Journal of Neuroscience 30*, 25 (2010), 8400–8410.

[91] Legenstein, R., Pecevski, D., and Maass, W. A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Comput Biol 4*, 10 (2008), e1000180.

[92] Levy, W., and Steward, O. Temporal contiguity requirements for long-term associative potentiation/depression in the hippocampus. *Neuroscience 8*, 4 (1983), 791–797.

[93] Lewis, M. A., Hartmann, M. J., Etienne-Cummings, R., and Cohen, A. H. Control of a robot leg with an adaptive avlsi cpg chip. *Neurocomputing 38* (2001), 1409–1421.

[94] Lewis, M. A., Tenore, F., and Etienne-Cummings, R. Cpg design using inhibitory networks. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on* (2005), IEEE, pp. 3682–3687.

[95] Linares-Barranco, B., Sánchez-Sinencio, E., Rodríguez-Vázquez, A., and Huertas, J. L. A cmos implementation of fitzhugh-nagumo neuron model. *Solid-State Circuits, IEEE Journal of 26*, 7 (1991), 956–965.

[96] Liu, B., and Frenzel, J. A cmos neuron for vlsi circuit implementation of pulsed neural networks. In *IECON 02 [Industrial Electronics Society, IEEE 2002 28th Annual Conference of the]* (Nov 2002), vol. 4, pp. 3182–3185 vol.4.

[97] Maas, A. L., Le, Q. V., Vinyals, O., Nguyen, P., and Ng, A. Y. Recurrent neural networks for noise reduction in robust asr.

[98] Maass, W. Liquid state machines: motivation, theory, and applications. *Computability in context: computation and logic in the real world*, 275–296.

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

[100] Maass, W. Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons. *Institute for Theoretical Computer Science. Technische Universitaet Graz. Graz, Austria, Technical Report TR-1999-037.[Online]. Available: http://www. igi. tugraz. at/psfiles/90. pdf* (1999).

[101] Maass, W., Schnitger, G., and Sontag, E. On the computational power of sigmoid versus boolean threshold circuits. In *Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on* (Oct 1991), pp. 767–776.

[102] Maass, W., Schnitger, G., and Sontag, E. D. On the computational power of sigmoid versus boolean threshold circuits. In *Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on* (1991), IEEE, pp. 767–776.

[103] Maher, M., Pine, J., Wright, J., and Tai, Y.-C. The neurochip: a new multielectrode device for stimulating and recording from cultured neurons. *Journal of neuroscience methods 87*, 1 (1999), 45–56.

[104] Maheswaranathan, N., Ferrari, S., VanDongen, A. M., and Henriquez, C. S. Emergent bursting and synchrony in computer simulations of neuronal cultures. *Frontiers in Computational Neuroscience 6* (2012).

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

[106] Martin, J. P., Guo, P., Mu, L., Harley, C. M., and Ritzmann, R. E. Central-complex control of movement in the freely walking cockroach. *Current Biology 25*, 21 (2015), 2795–2803.

[107] Masquelier, T., Guyonneau, R., and Thorpe, S. J. Competitive stdp-based spike pattern learning. *Neural computation 21*, 5 (2009), 1259–1276.

[108] Masquelier, T., and Thorpe, S. J. Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Comput Biol 3*, 2 (2007), e31.

[109] Matarić, M. J. Behavior-based robotics as a tool for synthesis of artificial behavior and analysis of natural behavior. *Trends in cognitive sciences 2*, 3 (1998), 82–86.

[110] Mazumder, P., Hu, D., Ebong, I., Zhang, X., Xu, Z., and Ferrari, S. Digital implementation of a virtual insect trained by spike-timing dependent plasticity. *Integration, the VLSI Journal 54* (2016), 109–117.

[111] Mazumder, P., Hu, D., Ebong, I., Zhang, X., Xu, Z., and Ferrari, S. Digital implementation of a virtual insect trained by spike-timing dependent plasticity. *Integration, the VLSI Journal 54* (2016), 109–117.

[112] McKinstry, J. L., Edelman, G. M., and Krichmar, J. L. A cerebellar model for predictive motor control tested in a brain-based device. *Proceedings of the National Academy of Sciences of the United States of America 103*, 9 (2006), 3387–3392.

[113] Meliza, C. D., and Dan, Y. Receptive-field modification in rat visual cortex induced by paired visual stimulation and single-cell spiking. *Neuron 49*, 2 (2006), 183–189.

[114] Mohanty, S. K., and Lakshminarayananan, V. Optical techniques in optogenetics. *Journal of modern optics 62*, 12 (2015), 949–970.

[115] Mohemmed, A., Schliebs, S., Matsuda, S., and Kasabov, N. Span: Spike pattern association neuron for learning spatio-temporal spike patterns. *International Journal of Neural Systems 22*, 04 (2012), 1250012.

[116] Mohemmed, A., Schliebs, S., Matsuda, S., and Kasabov, N. Training spiking neural networks to associate spatio-temporal input–output spike patterns. *Neurocomputing 107* (2013), 3–10.

[117] Montague, P. R., Dayan, P., and Sejnowski, T. J. A framework for mesencephalic dopamine systems based on predictive hebbian learning. *The Journal of neuroscience 16*, 5 (1996), 1936–1947.

[118] Morris, C., and Lecar, H. Voltage oscillations in the barnacle giant muscle fiber. *Biophysical journal 35*, 1 (1981), 193.

[119] Morrison, A., Diesmann, M., and Gerstner, W. Phenomenological models of synaptic plasticity based on spike timing. *Biological cybernetics 98*, 6 (2008), 459–478.

[120] Nobukawa, S., and Nishimura, H. Enhancement of spike-timing-dependent plasticity in spiking neural systems with noise. *International journal of neural systems* (2015), 1550040.

[121] O'keefe, J., and Nadel, L. *The hippocampus as a cognitive map*. Oxford University Press, USA, 1978.

[122] Panzieri, S., Pascucci, F., and Ulivi, G. An outdoor navigation system using gps and inertial platform. *IEEE/ASME transactions on Mechatronics 7*, 2 (2002), 134–142.

[123] Paugam-Moisy, H., and Bohte, S. Computing with spiking neuron networks. In *Handbook of natural computing*. Springer, 2012, pp. 335–376.

[124] Pecevski, D., Maass, W., and Legenstein, R. A. Theoretical analysis of learning with reward-modulated spike-timing-dependent plasticity. In *Advances in Neural Information Processing Systems* (2007), pp. 881–888.

[125] Pennartz, C. Reinforcement learning by hebbian synapses with adaptive thresholds. *Neuroscience 81*, 2 (1997), 303–319.

[126] Pfister, J.-P., Barber, D., and Gerstner, W. Optimal hebbian learning: a probabilistic point of view. In *Artificial Neural Networks and Neural Information ProcessingICANN/ICONIP 2003*. Springer, 2003, pp. 92–98.

[127] Point, N. Optitrack. *Natural Point, Inc.,[Online]. Available: http://www. naturalpoint. com/optitrack/.[Accessed 22 2 2014]* (2011).

[128] Ponulak, F. Resume-new supervised learning method for spiking neural networks. *Institute of Control and Information Engineering, Poznan University of Technology.(Available online at: http://d1. cie. put. poznan. pl/˜ fp/research. html)* (2005).

[129] Porr, B., and Wörgötter, F. Isotropic sequence order learning. *Neural Computation 15*, 4 (2003), 831–864.

[130] Prezioso, M., Merrikh-Bayat, F., Hoskins, B., Adam, G., Likharev, K. K., and Strukov, D. B. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature 521*, 7550 (2015), 61–64.

[131] Rachmuth, G., Shouval, H. Z., Bear, M. F., and Poon, C.-S. A biophysically-based neuromorphic model of spike rate-and timing-dependent plasticity. *Proceedings of the National Academy of Sciences 108*, 49 (2011), E1266–E1274.

[132] Ritzmann, R., and Büschges, A. Adaptive motor behavior in insects. *Current opinion in neurobiology 17* (2007), 629–636.

[133] Ritzmann, R., and Büschges, A. Adaptive motor behavior in insects. *Current opinion in neurobiology 17* (2007), 629–636.

[134] Ritzmann, R. E., Harley, C. M., Daltorio, K. A., Tietz, B. R., Pollack, A. J., Bender, J. A., Guo, P., Horomanski, A. L., Kathman,

N. D., Nieuwoudt, C., et al. Deciding which way to go: how do insects alter movements to negotiate barriers? *Frontiers in neuroscience 6* (2012).

[135] Rowcliffe, P., and Feng, J. Training spiking neuronal networks with applications in engineering tasks. *Neural Networks, IEEE Transactions on 19*, 9 (2008), 1626–1640.

[136] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning internal representations by error-propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1*, vol. 1. MIT Press, Cambridge, MA, 1986, pp. 318–362.

[137] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Cognitive modeling 5*, 3 (1988), 1.

[138] Samson, R., Frank, M., and Fellous, J.-M. Computational models of reinforcement learning: the role of dopamine as a reward signal. *Cognitive neurodynamics 4*, 2 (2010), 91–105.

[139] Sanger, T. D. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural networks 2*, 6 (1989), 459–473.

[140] Schreiber, S., Fellous, J.-M., Whitmer, D., Tiesinga, P., and Sejnowski, T. J. A new correlation-based measure of spike timing reliability. *Neurocomputing 52* (2003), 925–931.

[141] Seltzer, M. L., Yu, D., and Wang, Y. An investigation of deep neural networks for noise robust speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on* (2013), IEEE, pp. 7398–7402.

[142] Semiconductor, F. Cmos, the ideal logic family.

[143] Seo, J.-s., Brezzo, B., Liu, Y., Parker, B. D., Esser, S. K., Montoye, R. K., Rajendran, B., Tierno, J. A., Chang, L., Modha, D. S., et al. A 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. In *Custom Integrated Circuits Conference (CICC), 2011 IEEE* (2011), IEEE, pp. 1–4.

[144] Serrano-Gotarredona, T., Masquelier, T., Prodromakis, T., Indiveri, G., and Linares-Barranco, B. Stdp and stdp variations with memristors for spiking neuromorphic learning systems.

[145] Shapero, S., Zhu, M., Hasler, J., and Rozell, C. Optimal sparse approximation with integrate and fire neurons. *International journal of neural systems 24*, 5 (2014).

[146] Shrestha, S. B., and Song, Q. Weight convergence of spikeprop and adaptive learning rate. In *Communication, Control, and Computing (Allerton), 2013 51st Annual Allerton Conference on* (2013), IEEE, pp. 506–511.

[147] Shrestha, S. B., and Song, Q. Adaptive learning rate of spikeprop based on weight convergence analysis. *Neural Networks 63* (2015), 185–198.

[148] Skiena, S. S. *The algorithm design manual: Text*, vol. 1. Springer Science & Business Media, 1998.

[149] Spears, W. M., De Jong, K. A., Bäck, T., Fogel, D. B., and De Garis, H. An overview of evolutionary computation. In *Machine Learning: ECML-93* (1993), Springer, pp. 442–459.

[150] Sporea, I., and Grüning, A. Supervised learning in multilayer spiking neural networks. *Neural computation 25*, 2 (2013), 473–509.

[151] Stein, R. B., Gossen, E. R., and Jones, K. E. Neuronal variability: noise or part of the signal? *Nature Reviews Neuroscience 6*, 5 (2005), 389–397.

[152] Still, S., Schölkopf, B., Hepp, K., and Douglas, R. J. Four-legged walking gait control using a neuromorphic chip interfaced to a support vector learning algorithm. In *NIPS* (2000), Citeseer, pp. 741–747.

[153] Strack, B., Jacobs, K. M., and Cios, K. J. Simulating vertical and horizontal inhibition with short-term dynamics in a multi-column multi-layer model of neocortex. *International journal of neural systems 24*, 05 (2014), 1440002.

[154] Strain, T. J., McDaid, L., Maguire, L. P., and McGinnity, T. M. A supervised stdp based training algorithm with dynamic threshold neurons. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on* (2006), IEEE, pp. 3409–3414.

[155] Taherkhani, A., Belatreche, A., Li, Y., and Maguire, L. P. Dl-resume: a delay learning-based remote supervised method for spiking neurons. *IEEE transactions on neural networks and learning systems 26*, 12 (2015), 3137–3149.

[156] Taherkhani, A., Belatreche, A., Li, Y., and Maguire, L. P. Multi-dl-resume: Multiple neurons delay learning remote supervised method. In *Neural Networks (IJCNN), 2015 International Joint Conference on* (2015), IEEE, pp. 1–7.

[157] Takase, H., Fujita, M., Kawanaka, H., Tsuruoka, S., Kita, H., and Hayashi, T. Obstacle to training spikeprop networkscause of surges in training process. In *2009 International Joint Conference on Neural Networks* (2009), IEEE, pp. 3062–3066.

[158] National Academy of Engineering. NAE grand challenges for engineering. `http://www.engineeringchallenges.org/9109.aspx`, 2015.

[159] Thompson, D. M. *Understanding audio: getting the most out of your project or professional recording studio*. Hal Leonard Corporation, 2005.

[160] Tiňo, P., and Mills, A. J. Learning beyond finite memory in recurrent networks of spiking neurons. *Neural computation 18*, 3 (2006), 591–613.

[161] Van Vreeswijk, C., Abbott, L., and Ermentrout, G. B. When inhibition not excitation synchronizes neural firing. *Journal of computational neuroscience 1*, 4 (1994), 313–321.

[162] Wade, J. J., McDaid, L. J., Santos, J. A., and Sayers, H. M. Swat: a spiking neural network training algorithm for classification problems. *Neural Networks, IEEE Transactions on 21*, 11 (2010), 1817–1830.

[163] Wagenaar, D. A., Pine, J., and Potter, S. M. Searching for plasticity in dissociated cortical cultures on multi-electrode arrays. *Journal of negative results in biomedicine 5*, 1 (2006), 1.

[164] Walter, F., Röhrbein, F., and Knoll, A. Neuromorphic implementations of neurobiological learning algorithms for spiking neural networks. *Neural Networks 72* (2015), 152–167.

[165] Wang, J., Belatreche, A., Maguire, L., and McGinnity, T. M. An online supervised learning method for spiking neural networks with adaptive structure. *Neurocomputing 144* (2014), 526–536.

[166] Wang, Z., Guo, L., and Adjouadi, M. A generalized leaky integrate-and-fire neuron model with fast implementation method. *International journal of neural systems 24*, 05 (2014), 1440004.

[167] Weidenbacher, U., and Neumann, H. Unsupervised learning of head pose through spike-timing dependent plasticity. In *International Tutorial and Research Workshop on Perception and Interactive Technologies for Speech-Based Systems* (2008), Springer, pp. 123–131.

[168] Wilson, M. A., and Bower, J. M. The simulation of large-scale neural networks. In *Methods in neuronal modeling* (1989), MIT Press, pp. 291–333.

[169] Xin, J., and Embrechts, M. J. Supervised learning with spiking neural networks. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on* (2001), vol. 3, IEEE, pp. 1772–1777.

[170] Yamazaki, T., and Tanaka, S. The cerebellum as a liquid state machine. *Neural Networks 20*, 3 (2007), 290–297.

[171] Yang, J. J., Strukov, D. B., and Stewart, D. R. Memristive devices for computing. *Nature nanotechnology 8*, 1 (2013), 13–24.

[172] Zaghloul, K. A., and Boahen, K. Optic nerve signals in a neuromorphic chip ii: Testing and results. *Biomedical Engineering, IEEE Transactions on 51*, 4 (2004), 667–675.

[173] Zemelman, B. V., Lee, G. A., Ng, M., and Miesenböck, G. Selective photostimulation of genetically charged neurons. *Neuron 33*, 1 (2002), 15–22.

[174] Zhang, F., Wang, L.-P., Boyden, E. S., and Deisseroth, K. Channelrhodopsin-2 and optical control of excitable cells. *Nature methods 3*, 10 (2006), 785–792.

[175] Zhang, G., Rong, H., Neri, F., and Pérez-Jiménez, M. J. An optimization spiking neural P system for approximately solving combinatorial optimization problems. *International journal of neural systems 24*, 05 (2014), 1440006.

[176] Zhang, X. Snn-controlled insect simulation in virtual reality modeling language (vrml). `https://youtu.be/J31QcxCNuiY`, 2016.

[177] Zhang, X., Foderaro, G., Henriquez, C., and Ferrari, S. A scalable weight-free learning algorithm for regulatory control of cell activity in spiking neuronal networks. *International Journal on Neural systems* (2017).

[178] Zhang, X., Foderaro, G., Henriquez, C., VanDongen, A., and Ferrari, S. A radial basis function spike model for indirect learning via integrate-and-fire sampling and reconstruction techniques. *Advances in Artificial Neural Systems 2012* (2012), 10.

[179] Zhang, X., Xu, Z., Henriquez, C., and Ferrari, S. Spike-based indirect training of a spiking neural network-controlled virtual insect. In *Decision and Control (CDC), 2013 52th IEEE Annual Conference on* (2013), IEEE , pp. 6798–6805.

[180] Zhou, Y.-T., Chellappa, R., Vaid, A., and Jenkins, B. K. Image restoration using a neural network. *Acoustics, Speech and Signal Processing, IEEE Transactions on 36*, 7 (1988), 1141–1151.

[181] Zinkevich, M., Weimer, M., Li, L., and Smola, A. J. Parallelized stochastic gradient descent. In *Advances in neural information processing systems* (2010), pp. 2595–2603.

Biography

1. Xu Zhang
2. July 1988, China
3. Bachelor degree in Mechanical Engineering and Materials Science at University of Shanghai for Science and Technology, Master degree in Mechanical Engineering and Materials Science at Duke University, Master degree in Computer Science at Duke University.

1. Chinese National Scholarship, January 2007
2. First Class Honor, October 2009
3. University students of distinction, shanghai, May 2010
4. Shanghai Scholarship, December 2008