Indirect Training Algorithms for Spiking Neural Networks Controlled Virtual Insect Navigation

by

Xu Zhang

Department of Mechanical Engineering and Materials Science
Duke University

Date: _____

Approved:

_____
Silvia Ferrari, Supervisor

_____
Craig Henriquez

_____
Michael Zavlanos

_____
Brian Mann

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in the Department of Mechanical Engineering and Materials Science
in the Graduate School of Duke University
2015

Abstract

Indirect Training Algorithms for Spiking Neural Networks Controlled Virtual Insect Navigation

by

Xu Zhang

Department of Mechanical Engineering and Materials Science
Duke University

Date: _____

Approved:

_____
Silvia Ferrari, Supervisor

_____
Craig Henriquez

_____
Michael Zavlanos

_____
Brian Mann

An abstract of a thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in the Department of Mechanical Engineering and Materials Science
in the Graduate School of Duke University
2015

Abstract

Even though Artificial Neural Networks have been shown capable of solving many problems such as pattern recognition, classification, function approximation, clinics, robotics, they suffers intrinsic limitations, mainly for processing large amounts of data or for fast adaptation to a changing environment. Several characteristics, such as iterative learning algorithms or artificially designed neuron model and network architecture, are strongly restrictive compared with biological processing in natural neural networks. Spiking neural networks as the newest generation of neural models can overcome the weaknesses of ANNs. Because of the biologically realistic properties, the electrophysiological recordings of neural circuits can be compared to the outputs of the corresponding spiking neural network simulated on the computer, determining the plausibility of the starting hypothesis. Comparing with ANN, it is known that any function that can be computed by a sigmoidal neural network can also be computed by a small network of spiking neurons. In addition, for processing a large amount of data, SNNs can transmit and receive a large amount of data through the timing of the spikes and remarkably decrease the interactions load between neurons. This makes possible for very efficient parallel implementations.

Many training algorithms have been proposed for SNN training mainly based on the direct update of the synaptic plasticities or weights. However, the weights can not be changed directly and, instead, can be changed by the interactions of pre- and postsynaptic neural activities in many potential applications of adaptive

spiking neural networks, including neuroprosthetic devices and CMOS/memristor nanoscale neuromorphic chips. The efficiency of the bio-inspired, neuromorphic processing exposes the shortcomings of digital computing. After trained, the simulated neuromorphic model can be applied to speaker recognition, looming detection and temporal pattern matching. The properties of the neuromorphic chip enable it to solve the same problem while using fewer energies comparing with other hardware. The neuromorphic chips need applicable training methods that do not require direct manipulations of the connection strength.

Nowadays, thanks to fast improvements in hardware for neural stimulation and recording technologies, neurons in vivo and vitro can be controlled to fire precisely in milliseconds. These improvements enable the study on the link between synaptic-level and functional-level plasticity in the brain. However, existing training methods rely on learning rules for manipulating synaptic weights and on detailed knowledge of the network connectivity and synaptic strengths. New training algorithms that do not require the knowledge of the synaptic weights or connections are needed while they cannot require direct manipulations of the synaptic strength.

This thesis presents indirect training methods to train spiking neural networks, which can both modeling neuromorphic chips and biological neural networks in vivo, via extra stimulus without the knowledge of synaptic strengths and connections. The algorithms are based on the spike timing-dependent plasticity rule by controlling input spike trains. One of the algorithms minimizes the error between the synaptic weight and the optimal weight, by stimulating the input neuron with an adaptive pulse training determined by the gradient of the error function. Another algorithm uses numerical gradient of the output error with respect to the weight change to control the training stimulus, which are injected to the neural network for controlling a virtual insect for navigating and finding target in an unknown terrain. Finally, the newest algorithm uses indirect perturbation of the temporal differences between the

extra stimulus in order to train a large spiking neural network. The trained spiking neural network can control both a unicycle modeled virtual insect and a virtual insect moving in a tripod gait. The results show that these indirect training algorithms can train SNNs for solving control problems. In the thesis, the trained insect can find its target while avoiding obstacles in an unknown terrain. Future studies will focus on improving the insect's movement to using more complex locomotion model. The training algorithms will also be applied to biological neural networks and CMOS memristors. The trained neural networks will also be used for controlling flying microrobots.

Contents

List of Tables

List of Figures

List of Abbreviations and Symbols

Symbols

C_m	Membrane capacity.
I	Input current.
V	Voltage.
E_{Na}, E_K, E_L	Reversal Potentials of Sodium, Potassium and leaking.
g_{Na}, g_K, g_L	Sodium conductance, Potassium conductance and leaking conductance.
m, n, h	gating variables correlated to Sodium, Sodium and Potassium.
$\alpha_m, \alpha_n, \alpha_h$	voltage-dependent rate constant
$\beta_m, \beta_n, \beta_h$	voltage-dependent rate constant
$\tau_n(V), \tau_h(V)$	Time constants describing the dynamics of potassium channel and sodium channel.
V_{th}	Threshold of membrane potential.
t_i^f	Firing time of neuron i.
$\alpha(t - t_j^f)$	Alpha function for modeling the synaptic current.
q	: Total charge that is injected in a postsynaptic neuron via a synapse with weight 1.
g_{ij}	Synaptic conductance.
w_{ij}	Synaptic weight from neuron j to i.
$\tau_{rise}, \tau_{decay}$	Two time constants describing the rising rate and decay rate of the synaptic conductance.
τ_+, τ_-	Two time constant related to the STDP rule.

t_{pre}, t_{post}	Firing times of presynaptic neurons and postsynaptic neurons.
A_+, A_-	Maximum amplitude of weight change depend on the STDP rule.
Δt_{ij}	The firing time difference between pre- and postsynaptic neurons.
x_j, y_i	Two local variable for a low-pass filtered version of pre- and postsynaptic spike train.
$\mathcal{A}$	Object that is a compact subset of a workspace representing the physical characteristics of the virtual insect.
$\mathcal{W}$	Workspace where the insect navigate.
$\mathcal{B}_1, ..., \mathcal{B}_N$	Solid obstacles.
$\mathcal{S}, r$	Dashed circle and its radius.
$F_{\mathcal{W}}, O_{\mathcal{W}}$	Initial reference frame and its origin.
$F_{\mathcal{A}}, O_{\mathcal{A}}$	Moving reference frame and its origin.
x, y	Cartesian coordinates in $F_{\mathcal{W}}$
θ	Heading angle of the virtual insect.
$\mathcal{C}$	Insect's configuration space.
$\mathcal{T}_i$	Geometry of the target.
v_i	i_{th} motor speed.
L	The distance between two motors.
τ_{motor}	Time constant describing a gradual decay of motor speed.
S_m	Terrain sensor values.
γ	A scaling constant for the intensity of sensor inputs.
$M(x, y)$	Grayscale value at (x, y).
S_t	Target sensor values.
$P(x, y), T(x, y)$	Coordinates of the sensor and target, respectively.
g_L, g_R	Target sensor values.
d_L, d_R	Euclidean distances between the target position and the left and right target sensors.

λ	A constant value chosen heuristically to amplify the target sensor inputs differences between the left and right target sensor values.
α	The scaling factor of sensor inputs.
h_L, h_R	Left and right terrain sensors' stimuli.
σ	Roughness value.
f_o	Firing frequency of output neuron o.
C_L, C_R	Number of left and right output neurons.
ν_L, ν_R	Input Currents given to the left and right sets of CPG neurons.
α	A constant current input to the neurons in the locomotion system.
$s(t)$	RBF function given to LIF sampler
w^*	Desired weight.
ω_k, c_k, β_k	Height, center, width of the signal to IF neuron
I_{inj}	Extra stimulus to training neurons.
V_0	Resting potential of the neuron
θ	The potential difference between the resting potential and threshold.
R_m	Resistance of the membrane potential.
τ_m	Passive-membrane time constant.
τ_d	Conductance delay of the synaptic current from presynaptic neuron.
k_0, k_t	Index of the spike of the presynaptic neuron, which occurs after the previous spike of the second neuron; and the index of spike of the presynaptic neuron, which provoke the firing time of the second neuron.
θ	potential difference between V_{th} and V_0
τ_d	constant that represents the conduction delay of the synaptic current
$u_{k,s}(t)$	Averaging function.

t_k	Timings when IF sampler reaches its threshold.
θ	Threshold value for the IF sampler.
α	The leakage of the IF integrator.
a_E	Change in potential due to a single synaptic event, and depend on the weight of the synapse.
M	Number of presynaptic's spikes for the neural network with 2 or 3 neurons.
F_R, F_L	Expected firing frequencies of the left and right motor neurons.
S_{trL}, S_{tL}	Left terrain sensor value, the left target sensor value.
S_{trR}, S_{tR}	Right terrain sensor value and the right target sensor value.
α, γ	constants for scale the sensor values.
$f_i^p(t_l)$	The firing frequency of the neuron i during testing of case p at time interval $[t_{l-1}, t_l]$
δ	Error during testing of virtual insect neural network with 7 neurons.
d_{ji}	Time difference between the centers of the RBF input to neuron j and i.
c_i, c_j	Center of the RBF pulses to pre- and postsynaptic neuron.
λ	Learning rate for the 7-nodes neural network.
K	Number of either left or right motor neurons.
$z_i(t_r)$	Total number of spikes of output neuron i within the time interval $[t - t_r, t]$
k, η	constants for scale the sensor values for large neural network.
$\mathcal{N}$	A list of n possible ordered pairs selected to stimulate.
m	The number of training cases.
$p_{i,a}$	Temporal center of the a^{th} square pulse delivered to the i^{th} input neuron.
b_0	Initial temporal difference between the training square pulse inputs to the pre and postsynaptic neurons.

M	The number of training input pairings during each training epoch.
Δ_{abs}	an absolute refractory time of the neuron
$J(t)$	quadratic training-error function

Abbreviations

SNNs	Spiking Neural Networks
CMOS	Complementary metalCoxideCsemiconductor
RC circuit	A resistorCcapacitor circuit
HH Model	The first scientific model of a spiking neuron was proposed by Alan Lloyd Hodgkin and Andrew Huxley in 1952
ANNs	Artificial Neural Networks
STDP	Spiking Timing Dependent Plasticity
LIF	Leaky Integrate-and-Fire
ANNs	Artificial Neural Networks
RBFs	Gaussian Radial Basis Functions
XOR gate	a digital logic gate that implements an exclusive or
PSTH	Peri-Stimulus-Time Histogram
CISM	Neural Circuit Simulator

Acknowledgements

This work was supported by the National Science Foundation, under ECCS Grant 0925407 and Mechanical Engineering and Materials Science department(MEMS) at Duke University.

1

Background and Motivation

Spiking neural networks are computational models of living neurons comprehended of systems of differential equations that can duplicate some of the spike patterns and dynamics examined in living neuronal networks.Jack et al. (1975); Hodgkin and Huxley (1952a). They are also capable of simulating sigmoidal artificial neural networks (ANNs) and of solving small-dimensional nonlinear function approximation problems through reinforcement learning Maass (1997b); Pennartz (1997b); Ferrari et al. (2008a). Like all ANN learning methods, existing SNN training algorithms rely on the direct manipulation of the synaptic weights Ferrari et al. (2008a); Pennartz (1997b); Legenstein et al. (2005); Pfister et al. (2006); Florian (2007a); Wysoski et al. (2006). In other words, those learning rules ordinarily incorporate a weight-upgrade control by which the synaptic strengths are upgraded several times, based on the reinforcement signal or network performance. However, in many potential SNN applications, such as neuroprosthetic devices, light-sensitive neuronal networks grown in vitro, and CMOS/memristor nanoscale neuromorphic chips Jo et al. (2010), the synaptic weights cannot be changed directly by the learning algorithm. In these applications, the goal is to train a biological or artificial neural network to per-

form a complex function, for example processing an auditory signal or restoring a cognitive function. In neuroprosthetic therapeutic implants, for instance, a manufactured device may comprise of a microelectrode array or integrated circuit that empowers real neurons through spike trains. Therefore, the device cannot directly change the synaptic efficacies of the biological neurons, as do existing SNN training algorithms. However, it is possible to stimulate a subset of neurons by controlled pulses of electrical currents. As an alternate case, light-sensitive neuronal systems growed in vitro can be likely stimulated through controlled light patterns that cause chosen neurons to fire at exact times to control plasticity. At the same time, their output is being recorded progressively using a multielectrode array (MEA). For this situation, a PC can be utilized to determine the desired stimulation patterns for an in-vitro neuronal network with arbitrary connectivity, created by culturing dissociated cortical neurons from embryonic day E18 rat brain Van De Ven et al. (2005). Thus, the cultures may be utilized to check biophysical models of the mechanisms by which biological neuronal systems execute the control and storage of data by means of temporal coding. In this case, the actual connectivities and synaptic plasticities are regularly unknown and cannot be directly controlled as needed by existing SNN learning algorithms. This thesis presents indirect learning algorithms that assume synaptic weights cannot be updated or manipulated. The learning algorithms change the SNN weights by modulating spike timing dependent plasticity (STDP) through controlled input spike trains.

1.1 CMOS and Memristor

Complementary metal-oxide-semiconductor (CMOS) is a technology for constructing integrated circuits, which is applied to microprocessors, microcontrollers, static RAM, and other digital logic circuits. Low power requirements, high speed are the main strengths of CMOS technology. Improvements on CMOS Very-Large-Scale-

Integration reduce transistors to a smaller size. Howevers, scaling the transistors will not work forever because the CMOS circuits have process limitations on the design such as lithography or the area constraints Kuhn (2009); Wong et al. (1999).

Therefore, memristor becomes an encouraging technology to conquer these limitations Strukov et al. (2008). A memristor can be smaller than 10nm while offers high speed, lower space and non-volatility features Yakopcic et al. (2010). Additionally, Memristor is suitable for fabrication with CMOS for hybrid CMOS-Memristor based circuits. Memristive circuits are applicable in many fields such as the logic gate Borghetti et al. (2010), neuromorphic systems Kim et al. (2012), and analog circuits Shin et al. (2011). The improving transistor integration density in CMOS provides a good simulation of neural networks. The use of CMOS/memristors will enable an approximated synaptic plasticity, device density, scalability, and fault tolerance inherent in biological neural networks. In general, CMOS/memristor-devices have the potential for creation of adaptive, neuromorphic sensorimotor circuits for intelligent robots and neuroprosthetic devices that can adaptively interact in uncertain environments. The ultimate objective is not just a virtual realization of the CMOS/memristor hardware but to solve issues related to CMOS and memristor integration and provide results relating to the effectiveness of such a computational platform.

1.2 Light-Sensitive Culture

Amid the earlier years, systems for optical control of neuronal action have been produced Zemelman et al. (2002); Banghart et al. (2004); Zhang et al. (2006). Optogenetics is the combination of genetics and optics to control well-defined events within specific cells of living tissue. The improvement of combined optical/genetic approaches now enables scientist to see genetically targeted neurons in animals and to track and manipulate biochemical events within targeted cell types. One direc-

tion of the improvement of optogenetic methodologies is to target genetically optical control of neural activity in behaving *Drosophila*.

Optical incitement of electrically volatile cells is better than established actuation by microelectrodes. The purpose behind this is because of its high temporal-spatial resolution. Optical stimulus can be attained by utilizing caged compound, e.g. confined ATP, confined Glutamate, whereby the substrates for depolarizing particle channels are conveyed to membranes and initiated by the beats of UV-light to the chemical photolabile cage in the micro and millisecond time scale. An enhanced methodology is the utilization of light-switches connected to particle channels, which permits to depolarize cells in a reversible way with high temporal and spatial resolution. These methods, called chemical genetics, are useful for neural cells in little creatures like drosophila and zebrafish. The finding of channelrhodopsin2 (ChR2) from the unicellular alga Chlamydomonas reinhardtii was the beginning stage of the optogenetic methodology. When transfected into mammalian cells and initiated by blue light ChR2 works as an inwardly rectifying channel, depolarizing the cells. Together with the light-initiated, hyperpolarizing Cl-pump Halorhodopsin from the archea Natronomonas pharaonis, the two form a perfect pair for the activation and inactivation of neural cells (at two separate wavelengths). ChR2 enacts the cells with blue light by depolarization though NphR inactivates the cells with yellow light by hyperpolarization of the cells. As opposed to the chemical approach the microbial rhodopsins needn't bother with any additional expansion of chemicals, in light of the fact that the light sensing cofactor retinal is delivered by the host cells and ties effectively to the opsins, structuring the rhodopsins. ChR2 and NphR can be expressed in a more specific way in certain cell types of the neural system by utilization of better tuned promoters and molecular biology protocol. Since ChR2 and NphR and their variations can be effortlessly expressed in neural cells or used to structure transgenic creatures, these microbial rhodopsins have long been looking for apparatuses

for neurological research.

In general, for an experiment *in-vivo*, there are five steps. First of all, piece together genetic construct with promoter to drive expression and gene encoding opsin. Secondly, the construct is inserted into a virus. The third step is to inject the virus into the animal brain; opsin is then expressed in targeted neurons. The fourth step is to insert 'optrode', fibre-optic cable plus electrode into mouse brain. Finally, the laser light of a specific wavelength opens ion channel in neurons, which can control neuron spikes.

For the implementation of optogenetics *in-vitro*, neural networks with random connectivity can be obtained through a method known as Banker Cultures Fletcher et al. (1991), which grows neurons on top of a monolayer of astrocytes. Individual neurons can be labeled and visualized by transfecting neurons with Green fluorescent protein (GFP) colors and by using fluorescence imaging. Through the MEA etched onto the glass bottom the firing behavior of the culture output neurons can be continuously recorded and processed by spike detection/sorting software, allowing the real-time computation of firing rates or integrated signals. Output signals from the cultures can then be fed into the computational models that, in turn, can optically stimulate the cultures.

A Multi-Channel System recording hardware and software can acquire the raw data continuously from neurons in each culture, corresponding to the channels of each turnkey MEA system. Each neural channel can be amplified, filtered, and digitized at 31.25 kHz. As shown in Boyden et al. (2005), this experimental setup allows to record the neuronal activity on millisecond timescales, at the resolution of single spikes. Therefore, it is possible to map inputs to outputs in a culture, e.g. for training, and to utilize the output spike trains in real time, e.g. for control.

1.3 Spiking Neural Network Models

Artificial neural networks are as of now turning into an old model inside machine learning; the first thoughts and models are more than fifty years of age. The original of artificial neural networks comprised of McCulloch-Pitts neurons Maass (1997a), a reasonably straightforward model: a neuron sends a binary "high" output if the total of its weighted input signals climbs over an edge value. Despite the fact that these neurons can just give digital output, they have been effectively used in ANN like multi-layer perceptrons and Hopfield nets. For instance, any function with Boolean output can be processed by a multilayer perceptron with a single hidden layer.

The second-generation neuron model does not utilize a step or threshold function to calculate their output signals, yet a continuous activation function, let them applicable to analog in- and output. Regularly used actuation functions are the sigmoid and hyperbolic tangent. Average cases of neural networks comprising of neurons of these sorts are feed-forward and recurrent neural networks. These are more capable than the previous generation: when a threshold function at the output layer of the network is added, they are universal for advanced calculations and do so with fewer neurons than the previous model Maass et al. (1991). Furthermore, they can estimate any analog function very well, making these networks universal for analog calculations.

The third-generation neural network model raises the level of natural realism by utilizing individual spikes. This model permits joining spatial-temporal data in communication and processing, in the same way as real neurons do Ferster and Spruston (1995). So as opposed to utilizing rate coding these neurons use spike coding; systems where neurons receive and send individual spikes, permitting multiplexing of data as frequency and magnitude of sound Gerstner et al. (1999). Revelations in the field of neurology have demonstrated that neurons in the cortex perform signal

processings at an extraordinary rate. Thorpe et al Thorpe et al. (2001) exhibited that people process and recognize visual info (i.e. facial distinguishment) in shorter than 100ms. It makes more than ten synaptic strides from the retina to the temporal lobe; this leaves around 10ms of integration time for every neuron. Such a time window is much excessively little to permit an averaging method like rate coding Gerstner et al. (1999). This does not imply that rate coding is not utilized, though when quickness is an issue spike coding method are favored Thorpe et al. (2001).

For picking a spiking neural network model, we must discover bargains between two apparently fundamentally prerequisites: The model of a neuron must be: 1) computationally simple, yet 2) ready for creating many firing patterns showed by real neurons. Utilizing biophysically precise Hodgkin−Huxley models is computationally restrictive, since we can reproduce just a modest bunch of neurons continuously.

The Hodgkin-Huxley model is the beginning stage for detailed neuron models that represent various particle channels, distinctive sorts of synapses, and the particular spatial geometry of an individual neuron. The Hodgkin-Huxley model is additionally an essential reference model for the derivation of simpler neuron models.

The behavior of high-dimensional nonlinear differential equations is hard to view - and significantly harder to dissect. Two-dimensional differential equations, then again, can be mulled over in a straightforward way by method for a phase plane analysis. A reduction of the four-dimensional model of Hodgkin and Huxley to a two-variable neuron model is accordingly profoundly alluring. Several two-dimensional models by reduction from HH model include Morris-Lecar model, Fitzhugh-Nagumo model, and Izhikevich model Morris and Lecar (1981); FitzHugh (1961); Izhikevich et al. (2003).

However, intricate conductance-based neuron models can replicate electrophysiological measures to a high level of accuracy, but since their complexity these models are hard to dissect. Consequently, basic phenomenological spiking neuron models are

profoundly prevalent for the analysis of neural coding, memory, and system elements.

The leaky integrate-and-fire (LIF) neuron is presumably the best-known illustration of a basic spiking neuron model. Generally speaking, the leaky integrate-and-fire model incorporate the nonlinear leaky integrate-and-fire model. All integrate-and-fire model neurons can either be activated by external stimulation or synaptic currents from presynaptic neurons. The integrate-and-fire neuron model has been created as a canonical model for spiking neurons because it can be easily studied analytically while in the meantime being sufficiently plausible to catch large portions of the essential characteristics of a neural system.

1.4 Neuromorphic Systems

Recently, although the digital computer is very powerful and can solve very complex problems, the power requirement associated with the computer as the problem becomes more complex becomes prohibitive. For example, the best supercomputer cannot compare with the human brain (weighing around 1.5 kg). The brain can deal with driving during rush hour traffic with a power budget of about 20 W. IBM state of the art supercomputer, on the other hand, weighing 227 metric tons and taking up 5500 ft2 of area, requires close to 3 MW to simulate a few seconds of rush hour driving.

The efficiency of the bio-inspired, neuromorphic processing exposes the shortcomings of digital computing. In 1980s, neuromorphic computing is a concept describing the use of very-large-scale-integration(VLSI) systems containing electronic analog circuits to mimic neuro-biological architectures in the nervous system. In recent times, the term neuromorphic has been used to describe analog, digital, and mixed-mode analog/digital VLSI and software systems that using models of neural systems (for perception, motor control, or multisensory integration).

Implementing neural systems are the only thought to be neuromorphic on the

condition that they process in real time. Ongoing operation is a normal prerequisite for sensory systems, and numerous neuromorphic systems have been focused on sensory systems. Underneath, researchers study especially visual, sound-related, olfactory and touching nerve neuromorphic systems.

In Mead and Mahowald (1988), a "silicon retina" executes both the transduction (a fundamental photoreceptor circuit that has a near-logarithmic reaction), and a flat resistive layer that models the external plexiform layer of the retina. The first retina had 48 by 48 pixels. The system gives an output that is the contrast between the fovea intensity and a weighted normal of the enclosing intensities, which is truly dissimilar to what happens on a computerized cam. The general impact is that the reaction to a static edge is a spatial derivative, rather like what happens in a natural retina. What's more, the reaction to a featureless surface is zero free of the luminance.

Numerous papers have built up the thoughts inside this paper. Better photoreceptors are proposed and examined. These are utilized to make retinas more contrast-sensitive Andreou et al. (1995). By performing extra processing at the silicon retina, one can make the system ready to model particular visual capacities. For instance, a period to crash detector Indiveri (1998), and a model of the fly rudimentary movement monitor has been created Harrison and Koch (1998), and a fly elementary motion detector model has been created Yang et al. (2006). One especially fascinating recent progress has been a system that distinguishes intensity changes in a general luminance free way. In addition, it transmits these serially utilizing AER, consequently empowering sensing of quickly changing visual scenes without the gigantic information rates inferred by the need to process entire systems Lichtsteiner et al. (2008).

In summary, after being trained by some learning algorithms, the neuromorphic model on a computer can be applied to speaker recognition, looming detection and

temporal pattern matching Esser et al. (2013). However, the connection strengths on the neuromorphic chip cannot be directly updated through those learning rules. Biological plausible training algorithms are needed for training the neuromorphic chip to solving control problems.

1.5 Training Methods for Spiking Neural Networks

The effectiveness of SNN training algorithms to date remain very limited compared to artificial neural networks and has yet to be demonstrated on challenging control problems. One of the main difficulties to overcome is that the response of an SNN is not available in closed-form and must be obtained numerically by solving a system of differential equations. Additionally, the complex spike patterns of SNNs need to be decoded into reduced-order continuous signals for control or to assess the system-level performance.

An early supervised learning algorithm for SNN training, called SpikeProp, is a modification of the backpropagation ANN training method based on gradient descent, and directly manipulates the network's synaptic weights Bohte et al. (2002). However, it is required that the neurons are defined such that they fire once per signal, which severely limits the algorithm's use for most spike patterns. Several backpropagation approaches have expanded the capabilities of this method to more diverse spike signals, such as those that are encoded and decoded using rate coding Rowcliffe and Feng (2008); Ponulak and Kasinski (2010); Fiete and Seung (2006); Burgsteiner (2006); Sporea and Grüning (2013). These methods allowed for SNNs to be applied to more problems, such as classification and simple control problems, but the network architectures and spike train encoding and decoding techniques are unrealistic for biological neuronal networks in nervous systems or neuronal cultures Pennartz (1997a). Other approaches have been presented that change the structure of the network by adding and removing neurons during training, which allows for

adaptation of the SNN to new data without retraining Pfister et al. (2003); Dora et al. (2014), but this is also far from a biologically-plausible method. Additionally, the SNN weights, delays, and neuron numbers can be designed directly through optimization using genetic algorithms or linear programming to match spike rasters or perform classification Saleh et al. (2014); Rostro-Gonzalez et al. (2012). These approaches can match specific spike patterns and sequences well but also cannot be applied to biological networks.

It has been shown through biological experiments and observations that biological network plasticity is driven by Hebbian learning mechanisms Markram et al. (1997); Dan and Poo (1992); Levy and Steward (1983); Meliza and Dan (2006), Wang et al. (2014), such as spike-timing-dependent plasticity. The synaptic efficacy changes are based on the temporal correlation of firing activity of a set of pre and postsynaptic neurons. Several supervised learning approaches based on the STDP mechanism have be developed that have been effectively applied to nonlinear function approximation and classification problems Pfister et al. (2003); Dan and Poo (1992). This is also true for other approaches that use reinforcement learning methods based on STDP, which have been successful in nonlinear function approximation and classification as well Pennartz (1997a),Legenstein et al. (2010); Ferrari et al. (2008b); Foderaro et al. (2010); Florian (2007b). The reinforcement learning approaches use a global reward to simulate a global neurochemical signal that influences the STDP rule that induces synaptic weight changes within the network. It has also been recently shown that spike driven synaptic plasticity (SDSP) can be used for pattern recognition if it is assumed that the synaptic weights are known Kasabov et al. (2013). However, it is not within the realm of current technological capabilities to measure the synaptic weights and synaptic connections of in vivo neural networks. The SDSP rule increases the synaptic weight slightly every time a new spike arrives at the synapse and decreases slowly when there is no spike.

Many of the existing algorithms listed above are capable of performing function approximation or classification well. However for an approach to be viable in the training of biological neuronal networks or memristor-based neuromorphic computer chips, an algorithm must not require the direct manipulation of synaptic weights or knowledge of those weights, since the technology does not exist to do so. Instead of directly changing the weights, approaches have been developed that uses only input signals to perform supervisory training on the SNN Zhang et al. (2013, 2012). By controlling the precise timing and magnitude of input training signals, the SNN is trained to approximate a function and has been shown to serve effectively as a controller in a robot navigation application.

2

Spiking Neural Network Model

In the case of modeling a biological neuron, physical analogs are used in place of abstractions such as weight and transfer function. The input signal to a neuron is defined by an ion current, through the neuron membrane that occurs when neurotransmitters lead to an activation of ion channels in the membrane. This is modeled by a physical time-dependent current $I(t)$. The concentration of charged ions — on either side of the cell membrane — determines a capacitance C_m. Finally, the response of this neuron to the signal is a change in voltage between the neuron and its environment. This voltage change sometimes causes an action potential. After many experiments on a giant axon of the squid, Hodgkin and Huxley measured and derives mathematical differential equations for describing the neural dynamics. The Hodgkin-Huxley model is the primary reference for deriving the simple neuron models. Several nonlinear ODEs below model the neurons,

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

where E_{Na}, E_K, and E_L are the reversal potentials. Reversal potentials and

conductances are determined according to the experiments Hodgkin and Huxley (1952b).

In Eqn. 2.1, m,n,h are gating variables. Their dynamics are based on the differential equations below,

$$\begin{aligned} \dot{m} &= \alpha_m(V)(1-m) - \beta_m(V)m \\ \dot{n} &= \alpha_n(V)(1-n) - \beta_n(V)n \\ \dot{h} &= \alpha_h(V)(1-h) - \beta_h(V)h \end{aligned} \tag{2.2}$$

where $\dot{m} = dm/dt$, m,n,h can be considered as the probabilities of opening of sodium channel, closing of sodium channel, opening of potassium channel. $\alpha_m, \alpha_n, \alpha_h, \beta_m, \beta_n, \beta_h$ are the voltage-dependent rate constant.

Each equation in Eqn. 2.2 can be rewritten in the form,

$$\dot{x} = -\frac{1}{\tau_x(V)}[x - x_0(V)] \tag{2.3}$$

where x stands for m, n, or h. For fixed voltage V, the variable x approaches the value $x_0(V)$ with a time constant $\tau_x(V)$. The asymptotic value $x_0(V)$ and the time constant $\tau_x(V)$ are given by the transformation $x_0(V) = \alpha_x(V)/[\alpha_x(V) + \beta_x(V)]$ and $\tau_x(V) = [\alpha_x(V) + \beta_x(V)]^{-1}$.

The behavior of this high-dimensional neural network model is hard to visualize and applied in detail while two-dimensional neural network models can be studied by a phase plane analysis and easier to apply. Therefore, a reduction from Hodgkin-Huxley model to low dimensional neuron model is especially useful.

For reduction, a fast gating variable m representing the probability of opening of the sodium channel is treated as an instantaneous variable. Therefore, the equation for describing the variable m's dynamics are eliminated. Secondly, two time constants describing the dynamics of potassium channel $\tau_n(V)$ and the sodium channel $\tau_h(V)$ are approximately the same. Therefore, two equations for describing the closing of

sodium channel and opening of potassium channel can be reduced to one equation. Therefore, the four dimensional neuron models can be reduced to two dimension models such as Morris-Lecar Model Morris and Lecar (1981) and FitzHugh-Nagumo Model FitzHugh (1961); Nagumo et al. (1962).

Hodgkin-Huxley model does not have a so-called firing threshold. However, in a certain range of membrane potential, the neuron is very sensitive to the input pulses for generating and action potential. Therefore, this property can be fairly approximated by defining a threshold. Whenever a neuron's membrane potential reaches a threshold value, the neuron spikes transmit signals to other neurons.

2.1 Integrate-and-Fire Model

A systematic method to dimension reduction can be found in Kepler et al. (1992). Abbott and Kepler (1990) describes further reduction from a two-dimensional model to an integrate and fire model. The LIF SNN also provides the highest computational efficacy compared to other neurons models. Dayan and Abbott (2003); Gerstner and Kistler (2002).

In Fig. 2.1, an action potential happened from the left neuron can transfer the signal through the synapse to the postsynaptic neuron on the right. In the dashed square, the current I can be both synaptic current and external current. It is divided into two input currents I_R and I_C. The first variable can be calculated by Ohm's law as $I_R = V/R$. I_C charges the capacitor C. Then capacitive current $I_C = CdV/dt$. Therefore,

$$I(t) = C_m \frac{dV}{dt} + \frac{V(t)}{R} \tag{2.4}$$

After we use time constant $\tau_m = RC$ and multiply both sides by R, the standard LIF equation is,

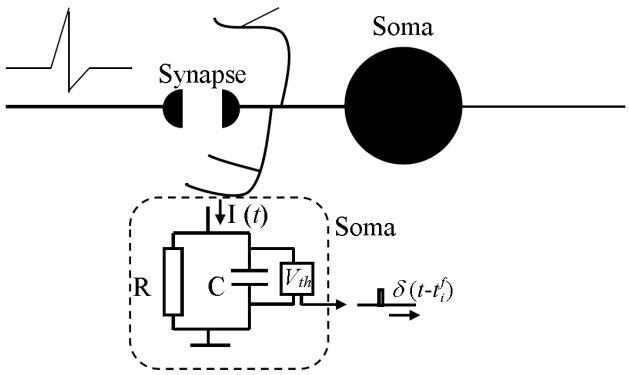


FIGURE 2.1: Diagram of the IF model. The basic circuit is the module inside the dashed square. The current I charges the circuit to increase the voltage $V(t)$ across the capacitance. Then it is compared with V_{th}. If the voltage is equal or over the threshold, the neuron spikes and generate an output pulse $\delta(t - t_i^f)$. The spike generates an input synaptic current to the postsynaptic neuron.

$$\tau_m \frac{dV(t)}{dt} = -V(t) + RI(t) \tag{2.5}$$

where $V(t)$ is the membrane potential, τ_m is the membrane time constant of the neuron. Whenever the $V(t)$ in Eqn. 2.5 increases above the threshold value V_{th}, it is reset to the resting potential. In Fig. 2.2, if the voltage is less than the resting potential shown as blue dash line, the dV will be positive for increasing the voltage. If the voltage is larger than the resting potential, dV will be negative to decrease the potential.

2.1.1 Synaptic Current

The inputs to the neurons can be extra stimulus via light or electrical signal and synaptic signals. Synaptic current is the main way for communications between neurons. After the neurons fires, the signal transfers from soma to another neuron's dendrite through the axon. The synaptic current can model either electrical or chemical signals. There are many different mathematical models for synaptic cur-

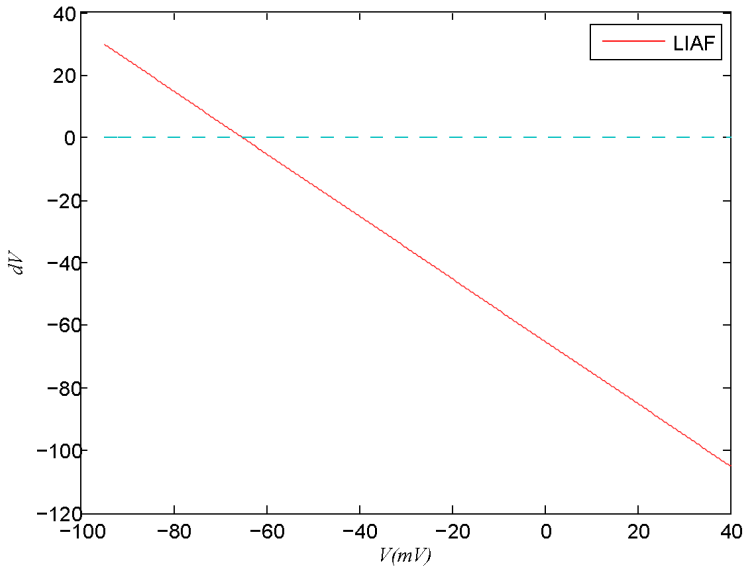


FIGURE 2.2: Red line is the membrane potential gradient with respect to the membrane potential. Blue dash line is the resting potential that equals to -65mv. No current input is given.

rents. Each neuron may receive synaptic currents from many neurons. Therefore, the synaptic current each neuron receives is,

$$I_i(t) = \sum_j w_{ij} \sum_f \alpha(t - t_j^f) \tag{2.6}$$

where w_{ij} is the synaptic weight between neuron i and j, f is the index of each spike, j is the presynaptic neurons index, t_j^f is the fth firing time of neuron j.

We use two different models including Dirac δ-pulse and α-function to model the $\alpha(\cdot)$ in Eqn. 2.6.

Dirac δ-pulse Synaptic Current : If a presynaptic neuron j fired a spike at t_j^f, the postsynaptic neuron receives current $\alpha(t - t_j^f)$. The Dirac δ-pulse models the current as $\alpha(s) = q\delta(s)$, where $s = t - t_j^f$. q is the total charge that is injected in a postsynaptic neuron via a synapse with weight $w_{ij} = 1$.

α-Function : According to Eqn. 2.6, the synaptic current from the presynaptic neurons j to postsynaptic neuron i can be modeled as,

$$I_{ij}(t) = -w_{ij}g_{ij}(s)[V(t) - E_{ij}], \quad where \quad \alpha(s) = -g_{ij}(s)[V(t) - E_{ij}] \tag{2.7}$$

where $g_{ij}(s)$ is the synaptic conductance, and E_{ij} is the reversal potential for the synapse. A realistic synaptic current model has a finite duration. Therefore, $g_{ij}(t)$ are modeled using this two-parameter alpha function for representing the rising and decay phases Schutter (2009),

$$g_{ij}(s) = h(e^{-\frac{s}{\tau_{delay}}} - e^{-\frac{s}{\tau_{rise}}}) \tag{2.8}$$

where, to ensure that the alpha function's amplitude equals 1, the normalization factor in (2.8) is defined as,

$$h = (-e^{-\frac{t_{peak}}{\tau_{rise}}} + e^{-\frac{t_{peak}}{\tau_{decay}}})^{-1} \tag{2.9}$$

and the conductance peaks at a time

$$t_{peak} = \frac{\tau_{decay}\ \tau_{rise}}{(\tau_{decay} - \tau_{rise})} \ln\left(\frac{\tau_{decay}}{\tau_{rise}}\right) \tag{2.10}$$

2.1.2 Refractory Period

In order to define the refractory period, HH model need to be mentioned again. In HH model, before the membrane potential reaches a threshold, the neuron's sodium channel and potassium channel are open. Because the sodium channels opens instantaneous, the membrane potential increases because sodium ions flux into the neuron. In the meantime, the potassium channel also starts to open slowly, and potassium ions flow out of the neurons. After the sodium channel shutdown, the potassium ions continue to flow out so that the membrane potential continues to decrease. Because the potassium channels activate longer, the membrane potential decreases to below

the resting potential, which is called hyperpolarizes. During this hyperpolarization and before the membrane potential goes back to its resting potential, it is almost impossible to let the neuron fire again. This time range is called the refractory period. The action potential and refractory period can be seen clearly in a simulated Hodgkin-Huxley model in Fig. 2.3.

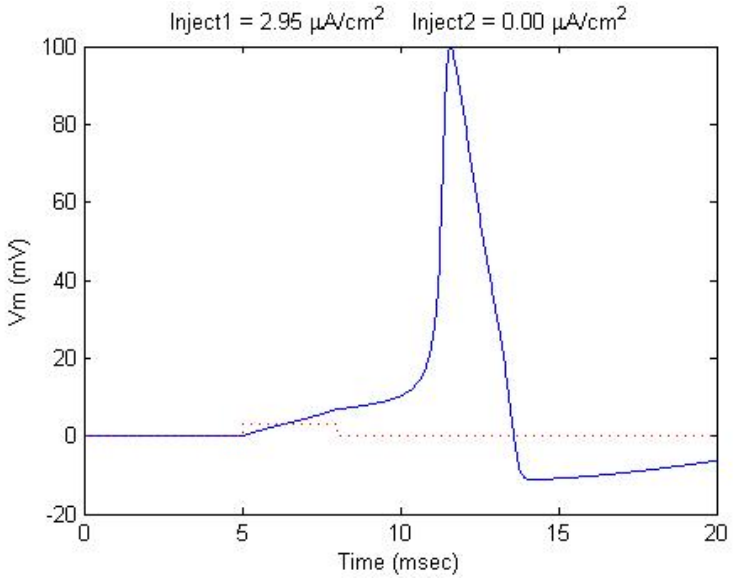


FIGURE 2.3: Action potential of the neuron modeled by the HH model.

The refractory period is after the membrane potential drops down below the resting potential 0 mv in Fig. 2.3. In Fig. 2.4, if the second current stimuli is during the refractory period, the neuron can't fire even with a larger stimulus.

However, if the currents stimulate the neuron after its previous refractory period, the neuron can fire again in Fig. 2.5.

In Fig. 2.6, LIF neuron model can also have this property by setting the membrane potential at the resting potential for a certain period time.

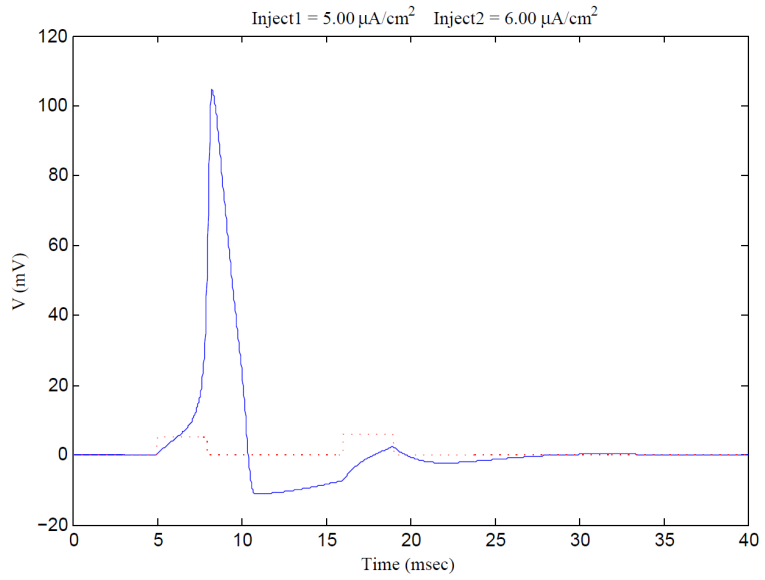


FIGURE 2.4: Neuron cannot fire during the refractory period.

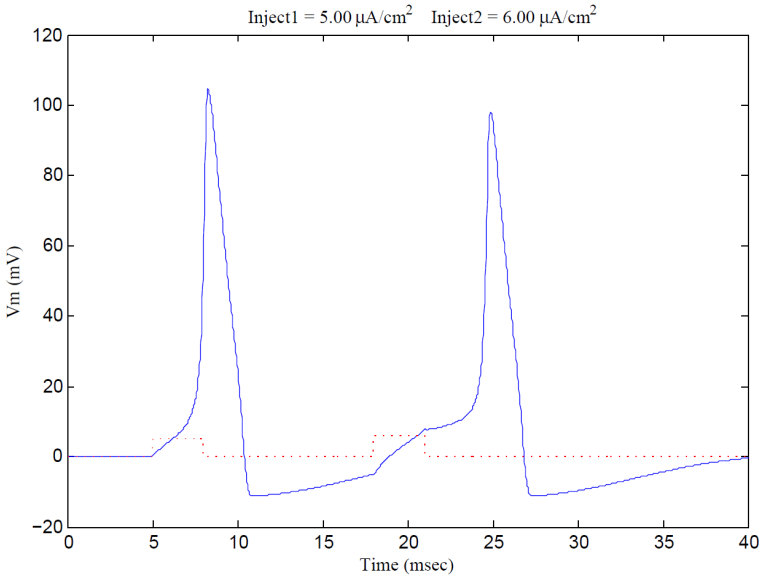


FIGURE 2.5: Neuron can fire after the refractory period.

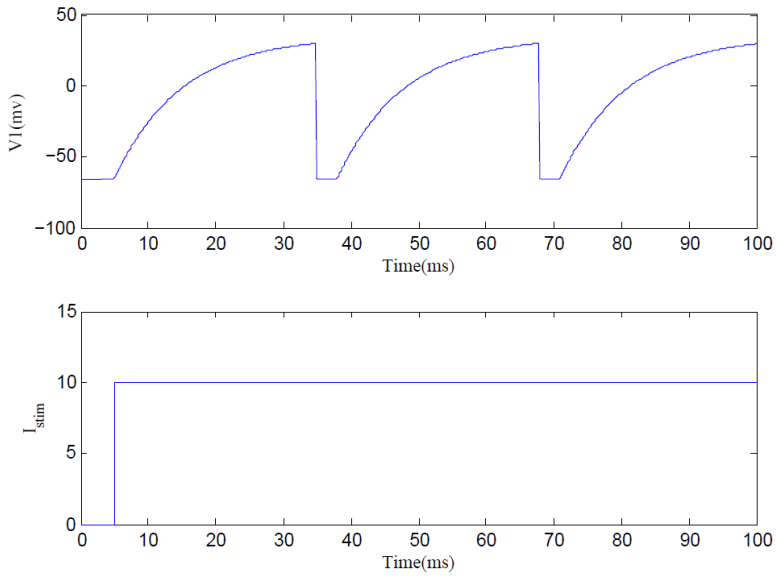


FIGURE 2.6: LIF neuron with a refractory period.

2.2 Hebbian Learning

In most neural network models, a synaptic strength $w_{i,j}$ determines the amplitude of the postsynaptic neuron's membrane potential response to the incoming spike. However, according to many biological experiments, the synaptic strength will change over time. Hebb proposes hypothesis that describes how the connection strength between two neurons will change according to its responses.

"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."

In Fig. 2.7, $w_{i,j}$ will only depend on the responses of neuron A and B. In other words, the synaptic weights change based on the firings of pre- and postsynaptic neurons called Hebbian learning.

After 20 years, the long-term synaptic weight changes are found during experiment Bliss and Lømo (1973); Bliss and Gardner-Medwin (1973). The results of

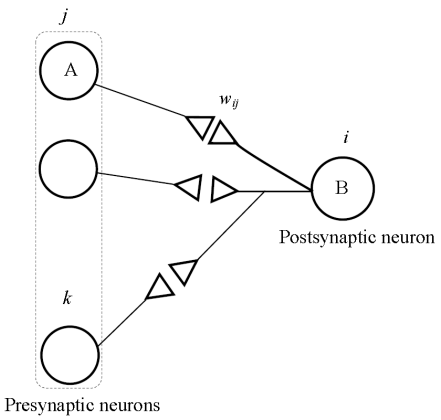


FIGURE 2.7: Change of synaptic weight $w_{i,j}$ only depends on the spikes of A and B.

those experiments are consistent with Hebbian's hypothesis; because the postsynaptic neuron is easier to be stimulated to fire after a period with many spikes of pre- and postsynaptic neurons. In 1970s, Long-term depression (LTD) was found among the synapses between the Schaffer collaterals and CA1 pyramidal cells in the hippocampus Albus (1971).

Even though the Hebbian's rule is proved, it does not explain how close the two neurons' spikes should be in order to increase the weight. Experiments start to study the weights change according to temporal differences between the pre- and postsynaptic neurons. According to experiments Bi and Poo (1998), the synaptic strength between two neurons can either increase or decrease according to the temporal differences of the firing times of pre- and postsynaptic neurons. This property is named Spike-Timing-Dependent Plasticity(STDP).

2.3 Spike Timing-Dependent Plasticity(STDP)

A persistent learning mechanism known as spike-timing-dependent plasticity, observed in biological neuronal networks, is used in this thesis to model synaptic plasticity in the LIF SNN. Synaptic plasticity refers to the mechanism by which the synaptic efficacies or *strengths* between neurons are modified over time, typically as

a result of the neuronal activity. These changes are known to be driven in part by the correlated activity of adjacently connected neurons. The directions and magnitudes of the changes are dependent on the relative timings of the presynaptic spike arrivals and postsynaptic firings.

If the presynaptic neuron fires shortly before the postsynaptic neuron, the strength of the connection will be increased. In contrast, if the presynaptic neuron fires after the postsynaptic neuron, the strength of the connection will be decreased as illustrated in Fig. 2.8. Two constants τ_+ and τ_- determine the ranges of the presynaptic to postsynaptic temporal differences over which synaptic strengthening and weakening occurs. Let the synaptic efficacy or strength be referred to as *weight*, and denoted by w. $A = [A_+, A_-]$ is the maximum amplitude of the change of weight due to a pair of spikes. t_{pre} and t_{post} denote firing times of the presynaptic neuron and the postsynaptic neuron respectively. Then, for each set of neighboring spikes, the weight adjustment is given by,

$$\Delta w = Ae^{[(t_{pre}-t_{post})\tau]} \tag{2.11}$$

such that, the synaptic weight increases when the postsynaptic spike is before the presynaptic spike, and the weight decreases when the opposite happens. The amplitude of the weight change Δw lessens as the time difference between the spikes becomes larger as is illustrated in Fig. 2.8. For simplicity, in this thesis, all changes in synaptic strengths are assumed to occur solely as a result of the spike-timing dependent plasticity mechanism. Different methods can be used to identify the spikes that give rise to the STDP mechanism. The *nearest-spike STDP model* Sjostrom et al. (2001) means that for every spike of the presynaptic neuron, its nearest postsynaptic spike is used to calculate the timing difference in (Eqn. 2.11), regardless of whether it takes place before or after the presynaptic firing. Then, from (Eqn. 2.11),

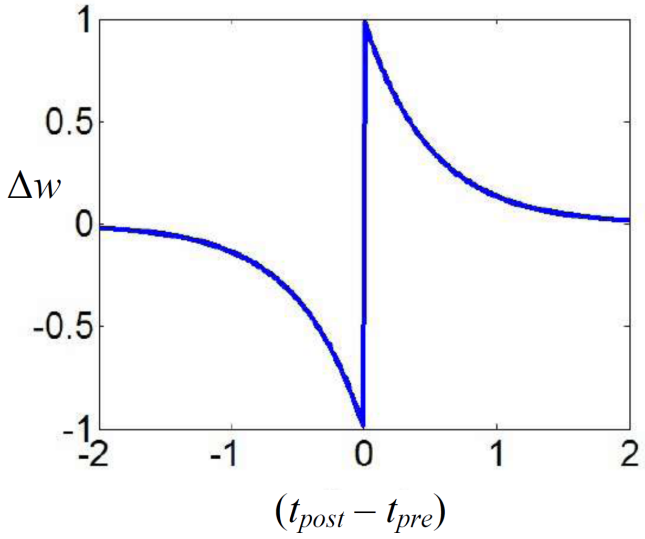


FIGURE 2.8: STDP term as a function of the time delay between the last spike of the postsynaptic neuron, and the presynaptic neuron.

the weight change due to one of the spikes of the ith neuron is modeled by the rule,

$$\Delta w_i(\Delta t_i) = \begin{cases} A_+ e^{\frac{\Delta t_i}{\tau_+}} & \text{if } \Delta t_i \leqslant 0 \\ -A_- e^{-\frac{\Delta t_i}{\tau_-}} & \text{if } \Delta t_i > 0 \end{cases} \tag{2.12}$$

where,

$$\Delta t_i = t_{1,i} - \underset{t_{2,k}}{\operatorname{argmin}} |t_{1,i} - t_{2,k}|. \tag{2.13}$$

Δt_i is the firing time difference between the two neurons, $t_{1,i}$ is the firing time of the presynaptic neuron, and $t_{2,i}$ is the firing time of the postsynaptic neuron. The constants used in this thesis are $\tau_+ = \tau_- = 20ms$, $A_+ = 0.05$, and $A_- = 1.05A_+$, where A_+ and A_- determine the maximum amounts of synaptic modification that occur when Δt is approximately zero Song et al. (2000).

From (Eqn. 2.12), the firing time of the postsynaptic neuron is chosen such that the absolute firing time difference between the presynaptic neuron and the postsynaptic neuron, $|t_{1,i} - t_{2,k}|$, is minimized. Therefore, the postsynaptic spike $t_{2,k}$ that is nearest the presynaptic firing time $t_{1,i}$ is used to calculate the firing time

difference Δt_i. The value of $t_{2,k}$ that minimizes $|t_{1,i} - t_{2,k}|$ can be found by comparing the firing time difference of the two neurons. In order to obtain an indirect learning method that does not rely on the direct manipulation of the synaptic weights, in this thesis, it is assumed that the weights can only be modified according to the STDP rule (Eqn. 2.12). Also, the training algorithm cannot specify any of the terms in (Eqn. 2.12) directly, but can only induce the firing times in (Eqn. 2.12) by stimulating the input neurons, e.g. through controlled pulses of electric voltages or blue light.

In this thesis, we also use all-to-all pairings on training neuron network to control virtual insect. For the all-to-all pairing, which means each presynaptic spike is compared with all previous postsynaptic spikes to calculate depression, and each postsynaptic spike is compared with all previous presynaptic spikes to calculate potentiation, the pair-based STDP rule in (2.12) can be numerically implemented in the LIF-SNN using two local variables, x_j for a low-pass filtered version of the presynaptic spike train, and y_i for the postsynaptic spike train Gerstner and Kistler (2006). Let us consider the synapse between neuron j and neuron i. Suppose that each spike from presynaptic neuron j contributes to a trace x_j at the synapse,

$$\frac{dx_j}{dt} = -\frac{x_j}{\tau_x} + \sum_{t_j^f} \delta(t - t_j^f) \tag{2.14}$$

where t_j^f denotes the firing times of the presynaptic neuron. In other words, the trace x_j is increased by an amount of one at t_j^f and decays with time constant τ_x afterwards. Similarly, each spike from postsynaptic neuron i contributes to a trace y_i,

$$\frac{dy_i}{dt} = -\frac{y_i}{\tau_y} + \sum_{t_i^f} \delta(t - t_i^f) \tag{2.15}$$

where t_i^f denotes the firing times of the postsynaptic neuron. When a presynaptic

spike occurs, the weight decreases proportionally to the momentary value of the postsynaptic trace y_i. Similarly, when a postsynaptic spike occurs, a potentiation of the weight is induced, such that:

$$\Delta w_{ij}(t_i^f) = D_+(w_{ij}) \cdot x_j(t_i^f) \tag{2.16}$$

$$\Delta w_{ij}(t_j^f) = -D_-(w_{ij}) \cdot y_i(t_j^f) \tag{2.17}$$

3

Benchmark Training Problem: Virtual Insect Control and Navigation

3.1 Unicycle Modeled Virtual Insect

The two different spike-based indirect training approaches presented in this paper is demonstrated on a path planning and control problem in which a virtual insect equipped with target and terrain sensors, processes environmental autonomously via SNNs. The physical characteristics of the virtual insect can be described by a rigid object $\mathcal{A}$ that is a compact subset of a workspace $\mathcal{W} \subset \mathbb{R}^2$. The workspace $\mathcal{W}$ can either contain smooth or rugged territory, with roughness and solid obstacles, denoted by $\mathcal{B}_1, ..., \mathcal{B}_N$, that must be avoided by $\mathcal{A}$. The virtual insect uses two antennas, shown in Fig. 3.1, where the dashed circle $\mathcal{S}$ of radius r represents the field of view of the left (target) antenna, which depends on the terrain roughness, and the red dot at the top of the red antenna is the field of view of the right (terrain) antenna. Using sensory information obtained by the antennas, the virtual insect must process the sensory inputs and adjust its current state according to the corresponding target movement.

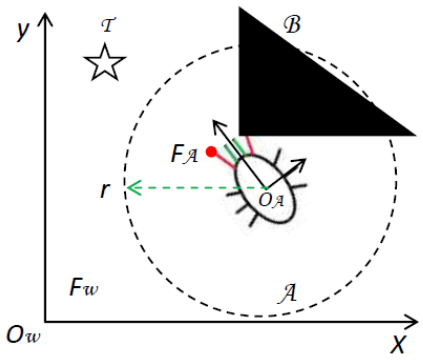


FIGURE 3.1: Insect geometry and workspace coordinates.

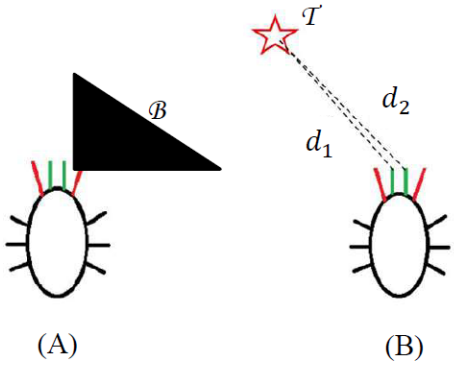


FIGURE 3.2: Insect terrain sensors (red antennas) (A), and target sensors (green antennas) (B).

The position and orientation of the insect with respect to $\mathcal{W}$ are defined with respect to an inertial reference frame $F_{\mathcal{W}}$ with origin $O_{\mathcal{W}}$ in $\mathcal{W}$. The insect's sensory inputs are defined with respect to a moving reference frame $F_{\mathcal{A}}$, embedded in $\mathcal{A}$, and with origin $O_{\mathcal{A}}$. It is assumed that both $\mathcal{A}$ and $\mathcal{S}$ are rigid objects, and that $\mathcal{S}$ has a fixed orientation and position with respect to $\mathcal{A}$. Let $q \in \mathbb{R}^3$ denote the configuration of the virtual insect, such that $q = [x\ y\ \theta]^T$, where x and y are the Cartesian coordinates in $F_{\mathcal{W}}$, and θ is the heading angle of the virtual insect. Then, $\mathcal{A}(q)$ denotes the compact subset of $\mathcal{W}$ that is occupied by $\mathcal{A}$ when the insect is at a configuration $q \in \mathcal{C}$, where $\mathcal{C}$ is the insect's configuration space. Similarly, the subset of $\mathcal{W}$ occupied by $\mathcal{S}$ at q can be denoted by $\mathcal{S}(q)$, and is the set of all the accessible sensor information that can be obtained by the insect when $\mathcal{T}_i \cap \mathcal{S}(q) = \varnothing$, where $\mathcal{T}_i$ is the geometry of the target.

The objective of the virtual insect is to reach the target $\mathcal{T}_i$, while avoiding rugged terrain in $\mathcal{W}$ by using visual and terrain information obtained from its antennas. Although the problem formulation and equations presented in this section are used to describe the insect motion, they are not used to design the insect SNN controller.

Instead, the SNN is trained using the spike-based indirect algorithm presented in Chapter 4, based on sensory inputs to which the insect responds at any time $t > t_0$.

The motion of the virtual insect is simulated using an adaptation of the unicycle robot that can more closely represent insect locomotion LaValle (2004),

$$\begin{cases} \dot{x} = v\ cos\theta \\ \dot{y} = v\ sin\theta \\ v = \frac{v_1+v_2}{2} \\ \dot{\theta} = \frac{v_2-v_1}{L} \\ \dot{v_i} = -\frac{v_i}{\tau_{motor}} + \eta \cdot H(t - t_f^i) \end{cases} \tag{3.1}$$

where v is the linear velocity, v_i is the i_{th} motor speed, $\dot{\theta}$ is the angular velocity, L is the distance between two motors, τ_{motor} is the time constant that results in a gradual decay of the motor speed following activation of the motor, t_f^i is the firing time of the output neuron i and $H(\cdot)$ is the Heaviside function which is scaled by a constant η.

3.1.1 Sensor Models of the Virtual Insect Trained by Gradient Method

The virtual insect uses terrain and target sensors, represented by the antennas in Fig. 3.2 in order to detect the roughness of the terrain, and the distances, d_1 and d_2, between the antennas and the target. The roughness of the terrain is represented by grayscale values, where 255 (white) is the flat region, and 0 (black) is the roughest region. Thus, the terrain sensor has an input,

$$S_m = \gamma |M(x, y) - 255| \tag{3.2}$$

where γ is a scaling constant for the intensity of sensor inputs, and $M(x, y)$ is the grayscale value at (x, y).

The two target (e.g. vision) sensors determine the position of the target $\mathcal{T}$ by calculating the Euclidean distance between each of the two (green) antennas, and

the target (star). Then, the sensory input from the target sensors is defined as,

$$S_t = \mu \| P(x, y) - T(x, y) \| \tag{3.3}$$

where μ is a scaling constant for the intensity of sensor inputs, and $P(x, y)$ and $T(x, y)$ are the coordinates of the sensor and target, respectively.

A gradient training method is applied to train a 7-nodes neural network with inhibitory and excitatory neurons for controlling a virtual insect. The SNN structure is shown in Fig. 3.3. The blue neurons are the four sensor neurons that receive the target and terrain sensor inputs. The two gray neurons are the output neurons that control the movement of the insect. Neuron 3 is the only inhibitory neurons serving as a computational neuron.

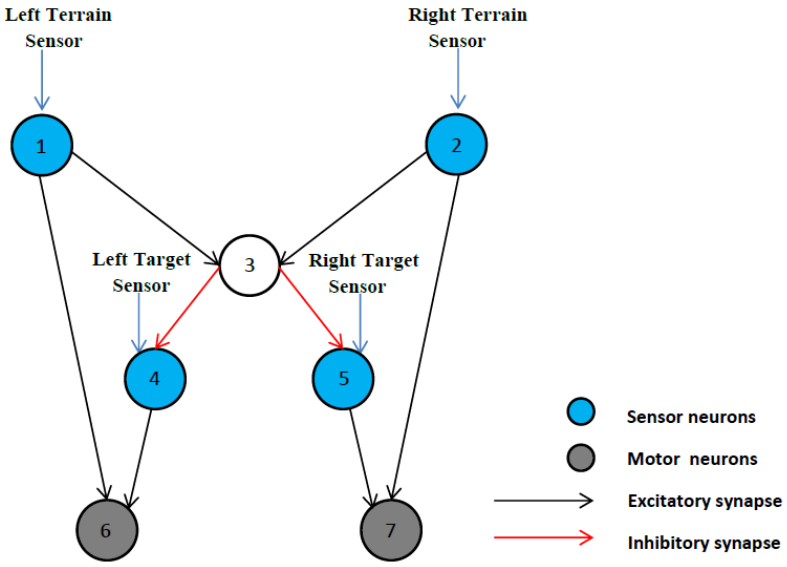


FIGURE 3.3: SNN architecture with 7 neurons.

3.1.2 Virtual Insects Trained by Weights Perturbation

For the perturbation training algorithm, in Fig. 3.2, the virtual insect used here has four total sensors of two types–two olfactory sensors to detect the insect's distance to the target, and two terrain sensors to feel the roughness of nearby terrain. The

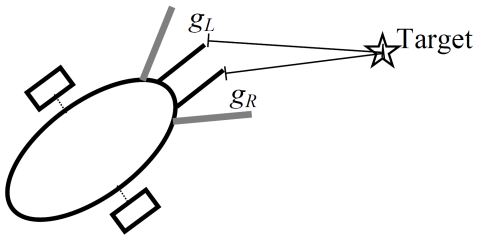


FIGURE 3.4: Target sensors and the distances between the sensors and the target.

measurements of these sensors are encoded into current stimuli for the sensory input neurons of the SNN. Each sensor has a fixed position on the insect geometry, $\mathcal{A}$, as illustrated in Fig. 3.1 as antennae, where the terrain sensors are represented by gray lines and the olfactory sensors by black lines. The measurements made by the sensors are assumed to occur at the ends of the antennae, and the magnitudes of the current stimuli are governed by sensor models described here.

The target sensors' stimuli, denoted by g_L and g_R for the left and right antennas, respectively, are modeled by,

$$\begin{aligned} g_L &= \alpha(d_L(x_L, y_L) + \lambda[d_L(x_L, y_L) - d_R(x_R, y_R)]) \\ g_R &= \alpha(d_R(x_R, y_R) + \lambda[d_L(x_R, y_R) - d_R(x_L, y_L)]) \end{aligned} \tag{3.4}$$

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. 3.4, 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.

In Fig. 3.5, the terrain sensors' stimuli, defined as h_L and h_R for the left and right antennae, respectively, are governed by,

$$\begin{aligned} h_L &= \alpha\gamma\tfrac{\sigma}{r(x_L, y_L)+1} \\ h_R &= \alpha\gamma\tfrac{\sigma}{r(x_R, y_R)+1} \end{aligned} \tag{3.5}$$

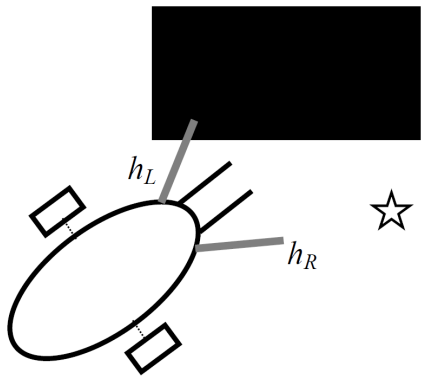


FIGURE 3.5: Terrain sensors and the obstacle touched by the left terrain sensor.

Table 3.1: Size of the SNN

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

where $\gamma = 0.1$ is a constant scalar chosen heuristically to bound the terrain sensor inputs, and σ denotes a roughness value such that when $r = \sigma$, the measured terrain is an obstacle.

The sensor data is applied to an input neuron layer of the SNN by injecting currents to the input neurons. The input neurons are a subset of neurons in the SNN that receive input signals. Within the input layer as illustrated in Fig. 3.6, the neurons are divided into four groups, where each group corresponds to and receives input signals from one of the four sensors at the same time. A 3-D structure of this SNN is shown in Fig. 4.15, where all neurons are randomly connected. The three layers are input layer, hidden layer or computational layer and output layer. The number of inhibitory and excitatory neurons in each layer is shown in Table. 3.1.

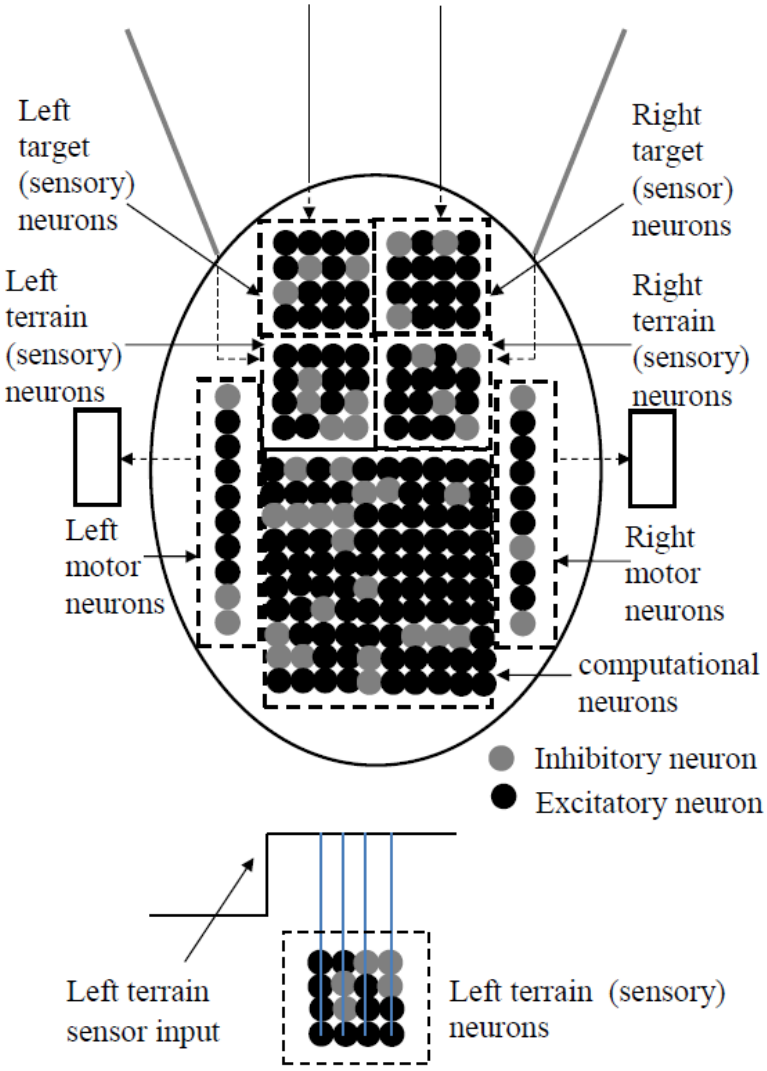


FIGURE 3.6: Sensor inputs to input sensory neurons.

3.2 Biologically Modeled Virtual Insect

For the biological model of virtual insect, the sensor models are the same as the unicycle model. However, the insect now is driven by another neural network controlling the insect's locomotion. In Fig. 3.9, the locomotion network receives the control input from the trained SNN as the Central Nervous System (CNS), which has 550 neurons in total. As the concept of mean firing rate has been used successfully for SNN signal decoding over the last 90 years, and previous works have

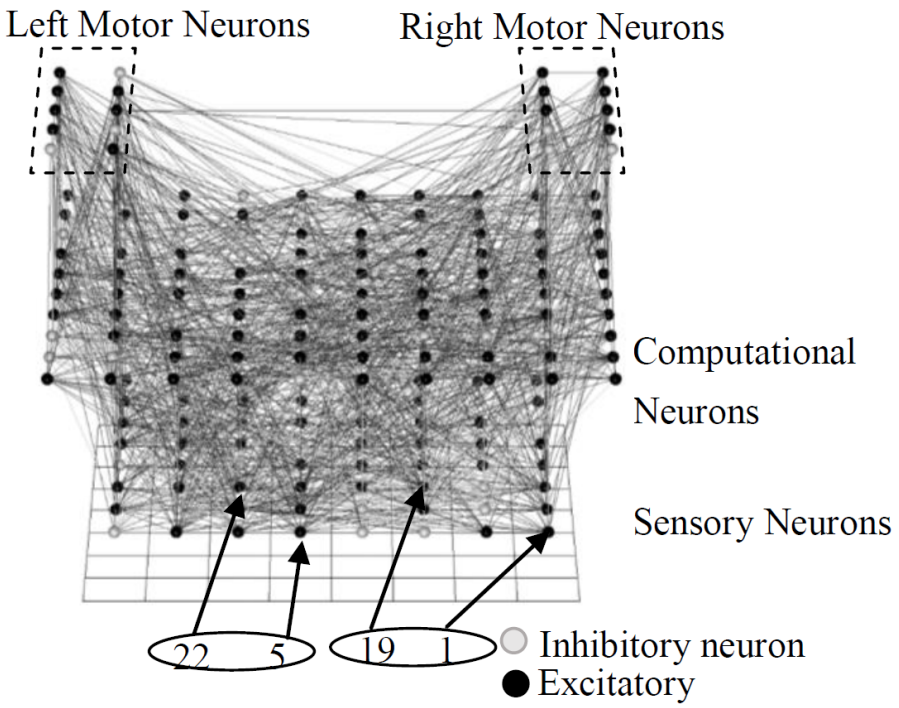


FIGURE 3.7: 3D structure of the SNN and two pairs of trained neurons. The training responses are shown in the Fig. 4.15.

shown a correlation between firing frequencies in muscle stretch receptor neural cells and applied force Mountcastle (1957); Hubel and Wiesel (1959), it is the decoding method used for control the biological locomotion model. The firing frequency of one neuron is defined by,

$$f_o = \frac{n(T)}{T} \tag{3.6}$$

where f_o is the firing frequency of the output neuron o within the time interval, T. Thus, the mean firing frequency is determined by,

$$f_L = \frac{\Sigma_{o=1,\ldots,C_L} f_o}{C_L} \tag{3.7}$$

$$f_R = \frac{\Sigma_{o=C_L+1,\ldots,C_L+C_R} f_o}{C_R} \tag{3.8}$$

where f_L, f_R are the average firing frequencies of the left and right output neurons in Fig. 3.9, C_L, C_R are the numbers of left and right output neurons. The average

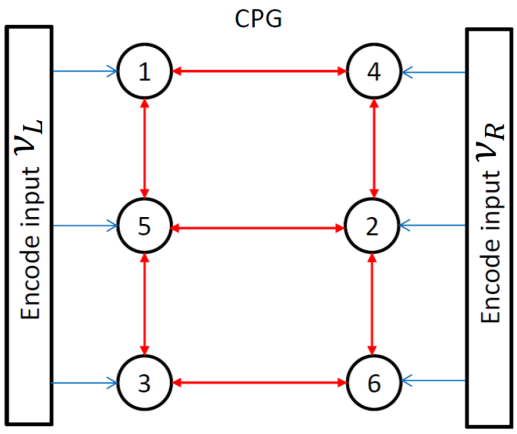


FIGURE 3.8: Inputs to CPG Neurons.

firing frequencies of the left and right output neurons are processed by a "winner gets all" decoder, where the output from the decoder to the CPG is based on the comparison between the left and right firing frequencies. If one side of the output has a higher average firing frequency, then the outputs to the CPG are constant for that side, while being zero for the other side. This is expressed in,

$$\nu_L = \pi \cdot H(f_R - f_L) \tag{3.9}$$

$$\nu_R = \pi \cdot H(f_L - f_R) \tag{3.10}$$

where ν_L, ν_R are the input currents given to the left and right sets of CPG neurons, f_L, f_R are the average firing frequencies of the left and right output neurons. $H(\cdot)$ is the Heaviside function, which determines which side will get the constant output, π , which is a constant current input to the neurons in the locomotion system.

In Fig. 3.8, ν_L is used for stimulating neurons 1, 5, 3, while ν_R is used to stimulate neurons 4, 5, 6. Stimulation of neurons 1, 5, 3 causes the left legs to flex, similarly neurons 4, 2, 6 flex the right legs. Additionally, neurons 4, 2, 6 are inhibited by the stimulation of neurons 1, 5, 3, keeping them extended.

The CPG neurons stimulated by the decoder output, are one of five sets of neurons that make up the locomotion neural network. The neurons model and network

connectivity among the CPG neurons, left and right motor neurons is discussed in **?**. All synapses in the CPG neurons are inhibitory, which is paramount for the plausible

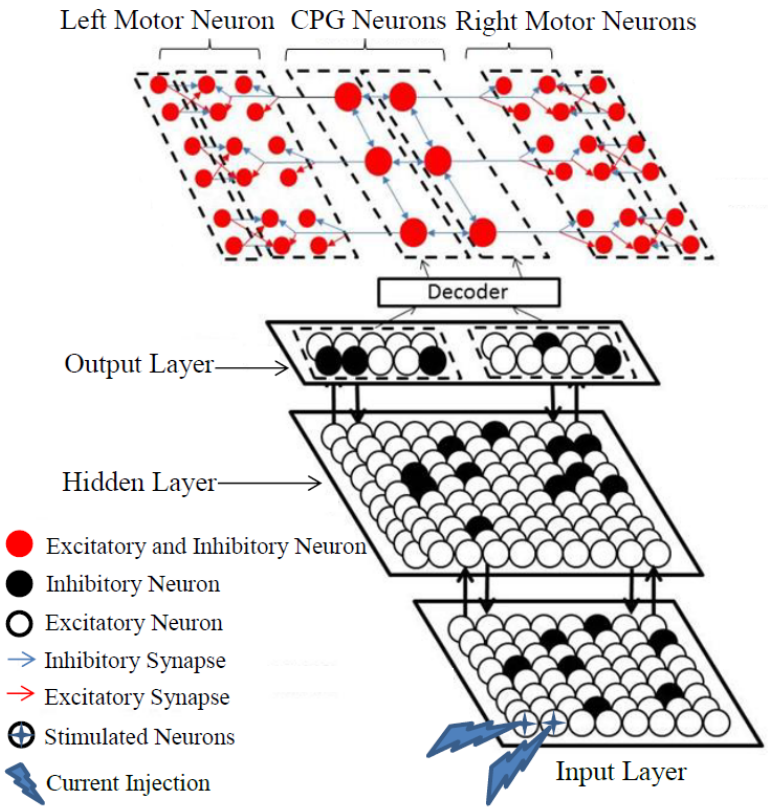


FIGURE 3.9: SNN architecture of a subset of neurons in a large neural network with 550 neurons.

mechanism of the insects' tripod leg movement. The inhibition of opposite legs, as detailed above ensures that three legs are on the ground at any time to stabilize the body of the insect, while the other three are executing a step.

4

Indirect Training Algorithms

4.1 Analytic Gradient Training Algorithm

Consider the two-node LIF SNN schematized in Fig. 4.1, modeled using the approach described in Section 2.1.1. $s(t)$ represents the input given to the LIF sampler. In Fig. 4.1, the SNN synaptic strength w_{21}, representing the synaptic efficacy for a pre-synaptic neuron, labeled by $i = 1$, and a post-synaptic neuron, labeled by $i = 2$, w_{21} cannot be modified directly by the training algorithm but can only change as a result of the STDP mechanism described in Section 2.3. In place of controlling w_{21}, the goal of the indirect training algorithm is to determine a spike train that can be used to stimulate the input neuron ($i = 1$) using I_{inj}, thereby inducing it to spike, such that the synaptic weight w_{21} changes from an initial (random) value to a desired value w^*.

For this purpose, we introduce a continuous spike model comprised of a superposition of Gaussian radial basis functions (RBFs),

$$s(t) = \sum_{k=1}^{N} w_k \exp[-\beta_k(||t - c_k||)^2] \tag{4.1}$$

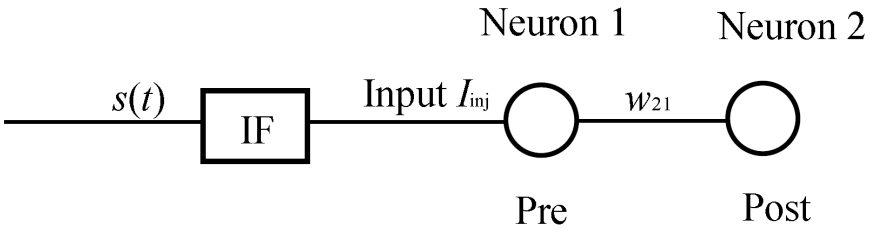


FIGURE 4.1: Model of two-node LIF spiking neural network.

where w_k determines the height of the kth RBF, which will decide the constant current input I_{inj}. β_k determines the width of the kth RBF, c_k determines the center of the kth RBF, where $k = 1, \ldots, N$, and N is the number of RBFs. Thus, the set of RBF adjustable parameters is $P = \{w_k, c_k, \beta_k \mid k = 1, \ldots, N\}$. A spike train can be obtained from the continuous spike model (Eqn. 4.1) by processing s using a suitable LIF sampler that outputs a square pulse function with the same heights, widths, and centers, as the RBF spike model (Eqn. 4.1).

In this two-node SNN model, there is no synaptic current sent to the first neuron because there are no pre-synaptic neurons connected to it. Rather, the input for the first neuron is the square pulse function provided by the output of the LIF sampler. Therefore, during one square pulse, the input, $I_{inj}(t)$, to neuron $i = 1$ can be viewed as a constant, which for our case is h, the height of the RBF. For a fixed threshold and a constant input, the time it takes for the first neuron to fire (or a spike to be generated) is,

$$T = -\tau_m \ln\left[1 - \frac{\zeta}{hR_m}\right] \quad (IR_m > \zeta) \tag{4.2}$$

where ζ is the potential difference between the spike generating threshold, V_{th}, and the resting potential, V_0, i.e., $\zeta = V_{th} - V_0$, h is the height of input square pulse, R_m is the resistance of the membrane, and τ_m is the passive-membrane time constant Burkitt (2006). Thus, from (Eqn. 4.2), the value of T can be controlled by adjusting h. We allow the first neuron to fire near the end of a square pulse by inputting a

spike sequence generated by an RBF model with a suitable width and height. Then, the firing times of the first neuron can be easily adjusted by altering the centers of the RBF.

In this simple example, it is assumed that the synapse is of the excitatory type. Therefore, (Eqn. 2.6) can be written as,

$$I_s(t) = C_m a_E \sum_{k=k_0}^{k_t} \delta(t - t_{1,k} - \tau_d) \tag{4.3}$$

where a_E is the change in potential due to a single spike from presynaptic excitatory neurons, τ_d is a known constant that represents the conduction delay of the synaptic current from neuron $i = 1$, and $t_{1,k}$ are the firing times of the first neuron. k_0 is the index of the spike of the first neuron, which occurs after the previous spike of the second neuron, k_t is the index of the spike of the first neuron, which provoke the firing time of the second neuron. The only input to the second neuron is the synaptic current defined in (Eqn. 4.3). Therefore, substituting (Eqn. 4.3) and (Eqn. 4.2) into (Eqn. 4.4),

$$C_m \frac{dv(t)}{dt} = I_{leak}(t) + I_s(t) + I_{inj}(t) \tag{4.4}$$

results in the following equation for the membrane potential:

$$C_m \frac{dv(t)}{dt} = -\frac{C_m}{\tau_m}[v(t) - V_0] + C_m a_E \sum_{k=k_0}^{k_t} \delta(t - t_{1,k} - \tau_d) \tag{4.5}$$

Solving (Eqn. 4.5) for the membrane potential of the second neuron provides the response of neuron $i = 2$,

$$v(t) = V_0 + \sum_{k=k_0}^{k_t} a_E \exp\left[-\frac{t - t_{1,k} - \tau_d}{\tau_m}\right] H(t - t_{1,k} - \tau_d) \tag{4.6}$$

where $H(\cdot)$ is the Heaviside function. Whenever the membrane potential in (Eqn. 4.6) exceeds the threshold V_{th}, the second neuron fires, and the membrane potential $v(t)$ is then set instantly equal to V_0. It follows that the firing times of the second neuron $t_{2,j}$ can be written as a function of the firing times of the first neuron,

$$t_{2,j} = (t_{1,k_t} + \tau_d)\ H\left\{ \sum_{k=k_0}^{k_t} a_E \exp\left[-\frac{t_{2,j} - t_{1,k} - \tau_d}{\tau_m} \right] H(t_{2,j} - t_{1,k} - \tau_d) - (V_{th} - V_0) \right\} \tag{4.7}$$

where j is the index of the firing times. In addition, the membrane potentials of the first neuron and the second neuron can be calculated using (Eqn. 4.6) and (Eqn. 4.2).

An example of the membrane potential time history for the two neurons is plotted in Fig. 4.2, where the square pulse function S used to stimulate the first neuron is plotted along with the neurons' membrane potentials v_1 and v_2. The red points denote the firing times and potentials for the two neurons. It can be seen from Fig. 4.2 that, with this input, the first neuron fires five times and the second neuron fires one time. In this SNN, the input to the second neuron is given only by the synaptic current caused by the firing of the first neuron. It can be seen that only when the second and third firing times of the first neuron are very close, the membrane potential of the second neuron increases over the threshold, causing it to fire. Whereas, the fourth and fifth firing times of the first neuron are too far apart to cause firing of the second neuron.

4.1.1 Deterministic Spike Model

The deterministic spike model consists of a continuous RBF model in the form (Eqn. 4.1) combined with an LIF sampler that converts the RBF into a square wave function. The LIF sampler developed in Feichtinger (2010) for the approximate reconstruction of bandlimited functions is adopted, which convert any continuous signal

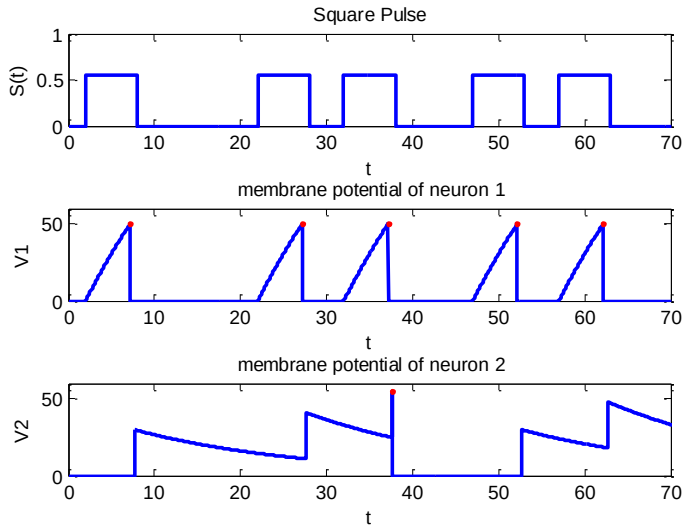


FIGURE 4.2: Membrane potential of the two neurons in the SNN in Fig. 4.1.

$f(t)$ into a square wave function by integrating it against an averaging function $u_{k,s}(t)$. The integrated result is compared to a positive threshold and a negative threshold, such that when either one of the two thresholds is reached, a pulse is generated at time $t_k = s$. The value of the integrator is then reset, and the process repeats. In this thesis, the averaging function $u_{k,s}(t)$ is chosen to be the exponential $e^{\varepsilon(t-s)}X_{[t_k,s]}$, where X_I is the characteristic function of I, and $\varepsilon > 0$ is a constant that models the leakage of the integrator, as due to practical implementations Feichtinger (2010). Then, the LIF sampler firing condition that generates the square wave (or sequence of pulses) is,

$$\pm\,\theta = \int_{t_k}^{t_{k+1}} f(t)e^{\varepsilon(t-t_{k+1})}dt \;\triangleq\; < f,\ u_k > \tag{4.8}$$

where t_k is the time instant of the sample, t_{k+1} is the next time instant of the sample, u_k is the averaging function, θ is the threshold value, and ε is the leakage of the integrator. The output of the LIF sampler can be expressed in terms of the time instants at which the integral reaches the threshold, $\{t_0, ..., t_n\}$, and by the samples

$q_1, ..., q_n$, defined as

$$q_j \triangleq \int_{t_j}^{t_{j-1}} f(x)e^{\varepsilon(x-t_j)}dx, \quad \text{for} \quad 1 \leqslant j \leqslant n \tag{4.9}$$

From (Eqn. 4.8), it follows that $|q_j| = \theta$. By adjusting the parameters of the LIF sampler, it is possible to convert the RBF signal in (Eqn. 4.1) to a square wave comprised of pulses with the same width, height and centers as the RBFs in (Eqn. 4.1). Thus, by using the RBF spike model in (Eqn. 4.8) with suitable widths and heights, it is possible to induce the first neuron to fire shortly before the end of each pulse of the square-pulse function (also considering the refractory period of the neuron). For simplicity, in this example, it is assumed that the heights w_k and widths β_k are known positive constants of equal magnitudes for all k. Then, the centers of the RBF comprise the set of adjustable parameters, $P = \{c_k \mid k = 1, \ldots, N\}$, to be optimized. The same approach can be easily extended to the case where all of the RBF parameters are adapted.

After adjusting the parameters of the RBF function, the firing times of the first neuron satisfy the constraints,

$$t_{1,k} \leqslant c_k + \frac{\beta}{2} \tag{4.10}$$

$$t_{1,k} + \Delta^{abs} \geqslant c_k + \frac{\beta}{2} \tag{4.11}$$

where Δ^{abs} is an absolute refractory time of the neuron. Then, the firing times of the first neuron can be written as a function of the RBF centers, c_k. By design, the first neuron fires after the same time interval, relative to each square pulse (Fig. 4.2), due to the chosen RBF heights and widths. Then, the firing times of the first neuron can be written as,

$$t_{1,k} = c_k - \frac{\beta}{2} + T \tag{4.12}$$

where, $t_{1,k}$ denotes the firing times of the first neuron, c_k are the centers of the RBF, β is the constant width of the RBF, and T is given by (Eqn. 4.2).

The RBF spike model is demonstrated in Fig. 4.3, where an example of RBF output obtained from (Eqn. 4.1) is plotted and, after being fed to the LIF sampler, produces a corresponding square wave which can be used as controlled pulse. It can be seen that the centers of the RBFs precisely determine the times at which a pulse occurs in the square wave and that the square wave maintains the desired constant width and magnitude for all k. The next subsection illustrates how, by adapting the continuous RBF spike model in (Eqn. 4.1), it is possible to minimize a desired error function, and train the synaptic weight of the SNN without directly manipulating it.

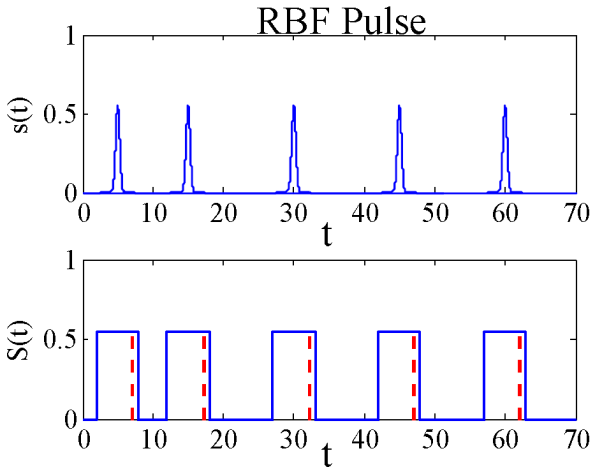


FIGURE 4.3: Deterministic Spike Model Signals

As explained in Section 2.3, it is assumed that the synaptic weight w_{21} can only be modified by controlling the activity of the SNN input neuron(s), with $i = 1$, and that it obeys that the nearest-spike STDP model in (Eqn. 2.12). Over time, the synaptic weight can repeatedly change and, therefore, its final value can be written as the sum of all incremental changes that have occurred over the time interval $[t_\iota, t_{\iota+1}]$,

$$w_{21}(t) = \prod_{i=1}^{M} w_0(1 + \Delta w_i) \tag{4.13}$$

where, for the two-neuron SNN in Fig. 4.1, $\Delta w_1, ..., \Delta w_M$ are due to M pairs of pre- and post-synaptic spikes that occur at any time $t \in [t_\iota, t_{\iota+1}]$. Every weight increment Δw_i is induced via STDP and, thus, obeys equation (Eqn. 2.12). Since the firing times in (Eqn. 2.12) depend on the centers of the RBF input, it follows that every weight increment Δw_i is a function of the RBF centers. In this example, the constraints (Eqn. 4.10)-(Eqn. 4.11) can be written as,

$$\frac{\beta}{2} < c_1, \quad c_N + \frac{\beta}{2} < t_f, \quad c_k + \beta < c_{k+1}, \quad \text{for} \quad k = 1, \ldots, N$$

where, from Fig. 4.3, $N = 5$.

A training-error function is defined in terms of the desired synaptic weight w^* and the actual value of the synaptic weight $w_{21}(t)$. While different forms of the error function may be used, the chosen form determines the complexity of the derivation of the training gradients. It was found that the most convenient form of training-error function can be derived from the ratio of the actual weight over the desired weight,

$$e(t) = \frac{w_{21}(t)}{w^*} \tag{4.14}$$

where $w_{21}(t)$ is the weight value at time $t \in [t_\iota,\ t_{\iota+1}]$, obtained from all previous spikes. The w^* can be viewed as the weight that leads to desired output $\hat{y}$ for a given input ξ. It can be seen that when w_{21} is equal to w^*, e is equal to one. Plugging (Eqn. 4.13) into (Eqn. 4.14), the weight ratio can be rewritten as,

$$e(t) = \frac{1}{w^*} \prod_{i=1}^{M} w_0(1 + \Delta w_i) \tag{4.15}$$

and the error between w_{21} and w^* can be minimized by minimizing the natural logarithm of the ratio (Eqn. 4.15),

$$E(t) \triangleq \ln\left[e(t)\right] = \ln(w_0) + \ln(1 + \Delta w_1) + \ldots + \ln(1 + \Delta w_M) - ln(w^*) \tag{4.16}$$

As is typical of all optimization problems, minimizing a quadratic form presents several advantages Stengel (1986). Therefore, at any time $t \in [t_\iota, t_{\iota+1}]$, the indirect training algorithm seeks to minimize the quadratic training-error function,

$$J(t) \triangleq E(t)^2 = \{\ln[e(t)]\}^2 = \{\ln(w_0) + \ln(1 + \Delta w_1) + \ldots + \ln(1 + \Delta w_M) - ln(w^*)\}^2 \tag{4.17}$$

Then, indirect training can be formulated as an unconstrained optimization problem in which J is to be minimized with respect to the RBF centers, or $P = \{c_k : \ c_k \in [t_\iota, t_{\iota+1}]\}$. Any gradient-based numerical optimization algorithm can be utilized for this purpose. The analytical form of the gradient $\partial J/\partial c_k$ depends on the form of the spike patterns. The following subsection derives this gradient analytically for one example of spike patterns and demonstrates that, using this gradient, the synaptic weight can be trained indirectly to meet the desired value w^* exactly. The same results were derived and demonstrated numerically for all other possible spike patterns, but they are omitted here for brevity. Another representation of error $e'(t)$ is defined below to make the convergence of the error more understandable in figures,

$$e'(t) = \frac{w_{21}(t)}{w^*} \tag{4.18}$$

4.1.2 Derivation of Gradient Equations for Indirect Training

Consider the case in which $M = 5$, $w^* = 1.72$, and $2a_E > V_{th} - V_0 > a_E$, which results in a two-node SNN (Fig. 4.1) in which neuron $i = 1$ must fire at least two times in order for neuron $i = 2$ to fire. An example of such a spike pattern is shown in Fig. 4.4, where N_1 and N_2 denote spike trains of neuron $i = 1$ and $i = 2$, respectively. In this case, the first neuron fires five times, and the second neuron fires two times. The firing time of the first neuron is controlled by the RBF spike model described in Section 4.1.1. In Fig. 4.4, $t_{i,k}$ denotes the k^{th} firing time of the i^{th} neuron with $k \in \mathcal{I}_i$, where $\mathcal{I}_i = \{k = 1, 2, ..., 5\}$ is an index set for the firing times, and where

i is the index labeling the neurons. In this example, the second neuron fires after $t_{1,3}$ and $t_{1,5}$, because $t_{1,2}$, $t_{1,3}$ and $t_{1,4}$, $t_{1,5}$ are close enough to cause the membrane potential of the second neuron, v_2, to exceed the threshold.

Initially, the synaptic weight is equal to 1.7. The weight change is discontinuous due to the discrete property of spikes. As shown in Fig. 4.4, the weight increases three times at $t_{1,3} + \tau_d$, decreases at $t_{1,4}$, and increases again at $t_{1,5} + \tau_d$. For this example, the synaptic weight changes in five increments

$$\Delta w_1 = A_+ \exp\left(\frac{t_{1,1}}{\tau_+}\right) \exp\left(\frac{-t_{1,3}}{\tau_+}\right) \exp\left(\frac{-\tau_d}{\tau_+}\right) \tag{4.19}$$

$$\Delta w_2 = A_+ \exp\left(\frac{t_{1,2}}{\tau_+}\right) \exp\left(\frac{-t_{1,3}}{\tau_+}\right) \exp\left(\frac{-\tau_d}{\tau_+}\right) \tag{4.20}$$

$$\Delta w_3 = A_+ \exp\left(\frac{-\tau_d}{\tau_+}\right) \tag{4.21}$$

$$\Delta w_4 = -A_- \exp\left(\frac{t_{1,3}}{\tau_-}\right) \exp\left(\frac{-t_{1,4}}{\tau_-}\right) \exp\left(\frac{\tau_d}{\tau_-}\right) \tag{4.22}$$

$$\Delta w_5 = A_+ \exp\left(\frac{-\tau_d}{\tau_+}\right) \tag{4.23}$$

When the equations above are substituted in (Eqn. 4.17), the training-error function can be written as,

$$\begin{aligned} J = & \left\{ \ln(w_0) + \ln\left[1 + A_+ \exp\left(\frac{t_{1,1}}{\tau_+}\right) \exp\left(\frac{-t_{1,3}}{\tau_+}\right) \exp\left(\frac{-\tau_d}{\tau_+}\right)\right] \right. \\ & + \ln\left[1 + A_+ \exp\left(\frac{t_{1,2}}{\tau_+}\right) \exp\left(\frac{-t_{1,3}}{\tau_+}\right) \exp\left(\frac{-\tau_d}{\tau_+}\right)\right] + \ln\left[1 + A_+ \exp\left(\frac{-\tau_d}{\tau_+}\right)\right] \\ & + \ln\left[1 - A_- \exp\left(\frac{t_{1,3}}{\tau_-}\right) \exp\left(\frac{-t_{1,4}}{\tau_-}\right) \exp\left(\frac{\tau_d}{\tau_-}\right)\right] \\ & \left. + \ln\left[1 + A_+ \exp\left(\frac{-\tau_d}{\tau_+}\right)\right] - \ln(w^*) \right\}^2 \end{aligned} \tag{4.24}$$

Then, the gradients of J with respect to the RBF centers are given by

$$\frac{\partial J}{\partial c_1} = \frac{\partial E^2}{\partial c_1} = \frac{\partial E^2}{\partial t_{1,1}} = 2E\left[\frac{\Delta w_1}{\tau_+\left(1+\Delta w_1\right)}\right] \tag{4.25}$$

$$\frac{\partial J}{\partial c_2} = \frac{\partial (E^2)}{\partial c_2} = \frac{\partial E^2}{\partial t_{1,2}} = 2E\left[\frac{\Delta w_2}{\tau_+\left(1+\Delta w_2\right)}\right] \tag{4.26}$$

$$\frac{\partial J}{\partial c_3} = \frac{\partial E^2}{\partial c_3} = \frac{\partial E^2}{\partial t_{1,3}} = \tag{4.27}$$

$$2E\left[\frac{-\Delta w_1}{\tau_+\left(1+\Delta w_1\right)} + \frac{-\Delta w_2}{\tau_+\left(1+\Delta w_2\right)} + \frac{\Delta w_4}{\tau_-\left(1+\Delta w_4\right)}\right] \tag{4.28}$$

$$\frac{\partial J}{\partial c_4} = \frac{\partial E^2}{\partial c_4} = \frac{\partial E^2}{\partial t_{1,4}} = 2E\left[\frac{-\Delta w_4}{\tau_-\left(1+\Delta w_4\right)}\right] \tag{4.29}$$

$$\frac{\partial J}{\partial c_5} = \frac{\partial E^2}{\partial c_5} = \frac{\partial E^2}{\partial t_{1,5}} = 0 \tag{4.30}$$

Using the above gradients, the optimal values of $c_1, \ldots, c_5$ can be obtained by minimizing J using a gradient-based numerical algorithm such as Newton's method.

An example of indirect learning algorithm implementation is shown in Fig. 4.4, where the RBF adjustable parameters (which, in this case, coincide with the firing times of neuron $i = 1$) are optimized to induce a change in the synaptic weight w_{21} from an initial value of 1.704, to a desired value $w^* = 1.72$. Another example, for which the gradient equations are omitted for brevity is shown in Figs. 4.5-4.7. In this case, the desired weight value $w^* = 2.8$ is far from the initial weight $w_{21}(t_\iota) = 3.5$, and thus $N = 43$ spikes are required to train the SNN. The optimal RBF input, and corresponding controlled pulse used to stimulate neuron $i = 1$, are shown in Figs. 4.5 and 4.6, respectively. The time history of the weight w_{21} is plotted in Fig. 4.7(a), along with the corresponding deviation from w^* plotted in Fig. 4.7(b).

A more general form of the gradient equations can be found by rewriting the

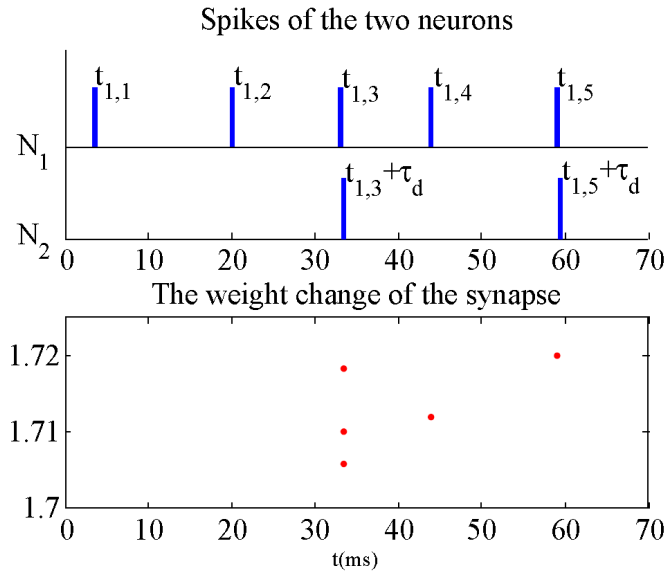


FIGURE 4.4: Optimized input spike train, and indirect weight changes brought about by STDP.

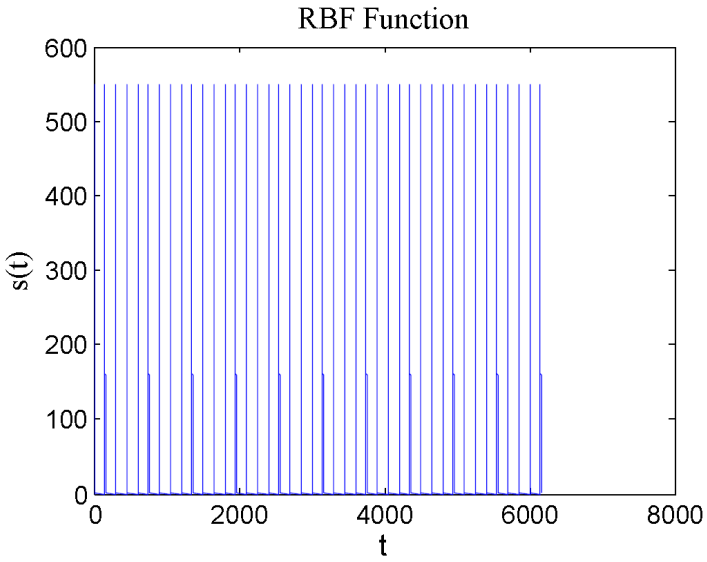


FIGURE 4.5: Optimized RBF spike model for the SNN in Fig. 4.1.

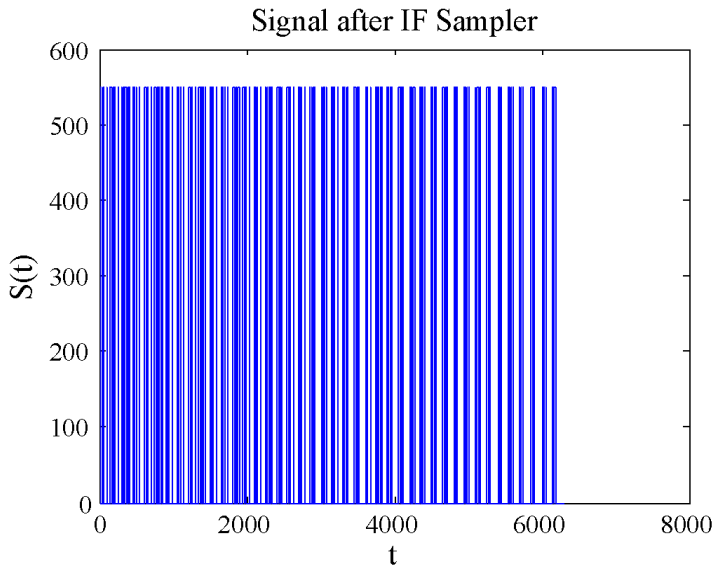


FIGURE 4.6: Square wave obtained by the LIF sampler, for the RBF spike model in Fig. 4.5.

response of the membrane potential for neuron $i = 2$ in (Eqn. 4.6), as follows,

$$v(t) = V_0 + \sum_{k=a}^{b} a_E \exp\left[-\frac{t - t_{1,k} - \tau_d}{\tau_m}\right] H(t - t_{1,k} - \tau_d) \tag{4.31}$$

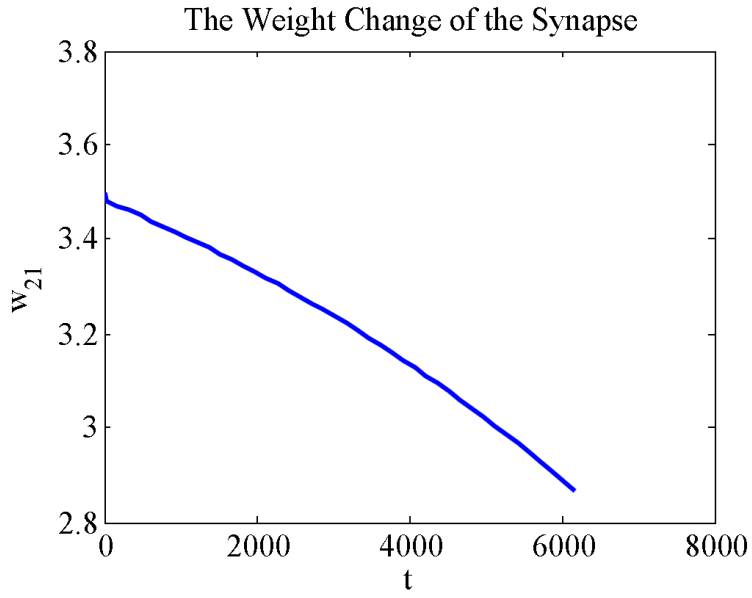
where $t_{1,k}$ denotes the kth firing time of neuron $i = 1$. Then, the gradients of the training objective function (Eqn. 4.17) with respect to the centers of the RBF spike model for $M = 5$ are given by the equations in the Appendix, which are obtained in terms of the two functions,

$$g_+(t_{1,i}, t_{1,j}) = A_+ \exp\left(\frac{t_{1,i}}{\tau_+}\right) \exp\left(\frac{-t_{1,j}}{\tau_+}\right) \exp\left(\frac{-\tau_d}{\tau_+}\right) \tag{4.32}$$

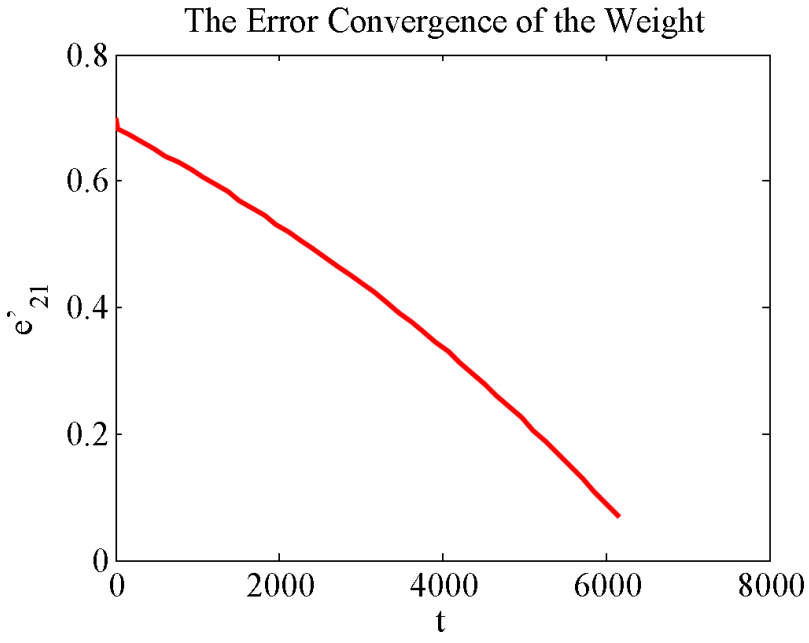
$$g_-(t_{1,i}, t_{1,j}) = -A_- \exp\left(\frac{t_{1,i}}{\tau_-}\right) \exp\left(\frac{-t_{1,j}}{\tau_-}\right) \exp\left(\frac{\tau_d}{\tau_-}\right) \tag{4.33}$$

where $t_{1,i}$, $t_{1,j}$ are two distinct firing times of neuron $i = 1$.

The methodology presented in Chap. 4 can also be extended to larger SNNs, although in this case it may be more convenient to compute the gradients of the



(a)



(b)

FIGURE 4.7: Indirect weight changes brought about by STDP, and the corresponding deviation from $w^* = 2.8$, for the SNN in Fig. 4.1 stimulated using the input spike train in Fig. 4.5.

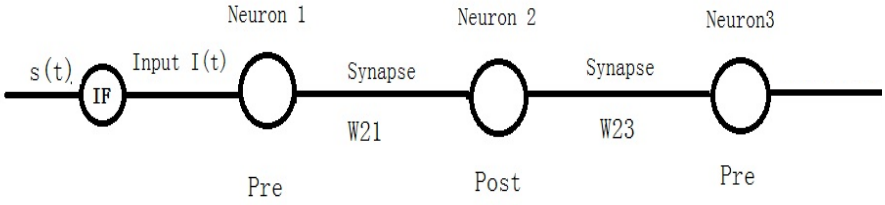


FIGURE 4.8: Model of three-node LIF spiking neural network.

objective function numerically. As an example, consider the three-neuron SNN in Fig. 4.8, and with two synaptic weights w_{21} and w_{23} that each obeys the STDP mechanism in (Eqn. 2.12). Suppose the desired values of the synaptic weights are $w_{21}^* = 4.1$ and $w_{23}^* = 2.8$. Using the indirect learning method presented in this thesis, and the gradient provided in the Appendix, a training-error function formulated in terms of the deviations of $w_{21}(t)$ and $w_{23}(t)$ from w_{21}^* and w_{23}^* respectively, can be minimized with respect to the parameters (centers) P of the RBF spike model (Eqn. 4.1).

Once the optimal RBF spike model, plotted in Fig. 4.9, is fed to the LIF sampler, the controlled pulse plotted in Fig. 4.10 is obtained and implemented via I_{inj}. The controlled pulse is thus used to stimulate neuron $i = 1$ at precise instants in time that correspond to centers of the optimal RBF spike model. Therefore, $w_{21}(t)$ can be converted to the desired value w_{21}^*. Then the same method will be implemented on the second neuron for changing $w_{23}(t)$ to w_{23}^*. The time-histories of the weight values, $w_{21}(t)$ and $w_{23}(t)$, obtained by the indirect training algorithm are plotted in Fig. 4.11(a). As is also shown by the corresponding training errors, plotted in Fig. 4.11(b), the SNN weights over time converge to the desired values, $w_{21}^* = 4.1$ and $w_{23}^* = 2.8$. These results demonstrate that, even for larger SNNs, the indirect training method presented in this thesis is capable of modifying synaptic weights until they meet their desired values, without direct manipulation. Since this indirect training algorithm only relies on modulating the activity of the input neurons, via

controlled input spike trains, it also has the potential of being realizable *in vitro* and *in silico*, to train biological neuronal networks, and in CMOS/memristor nanoscale chips, respectively.

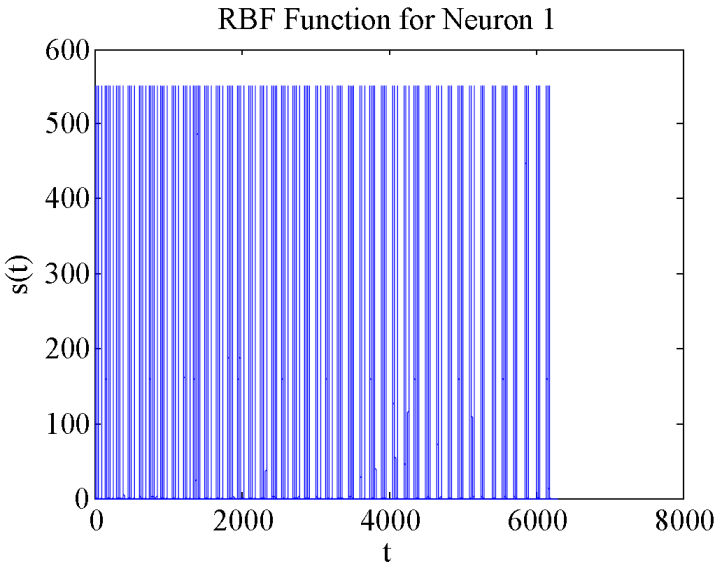


FIGURE 4.9: Optimized RBF spike model for the SNN in Fig. 4.8.

4.2 Numerical Indirect Gradient Method

4.2.1 Optimization of Radial Basis Function

The numerical Indirect Gradient Method is applied to the virtual insect problem in Section 3.1.1. If either the left or right terrain sensor detects the rough terrain, the insect adjusts its direction in order to avoid entering the respective location. This desired behavior can be mathematically formulated as follows,

$$\begin{pmatrix} F_L \\ F_R \end{pmatrix} = \begin{pmatrix} S_{trL} & S_{tL} \\ S_{trR} & S_{tR} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \tag{4.34}$$

where F_L and F_R are the expected firing frequencies of the left and right motor neurons; $S_{trL},\ S_{tL},\ S_{trR},\ S_{tR}$ are, respectively the left terrain sensor value, the left target sensor value, the right terrain sensor value, and the right target sensor value. The constants $a_1,\ a_2$ scale the sensor values, where $a_1 \gg a_2$ due to fact that inputs

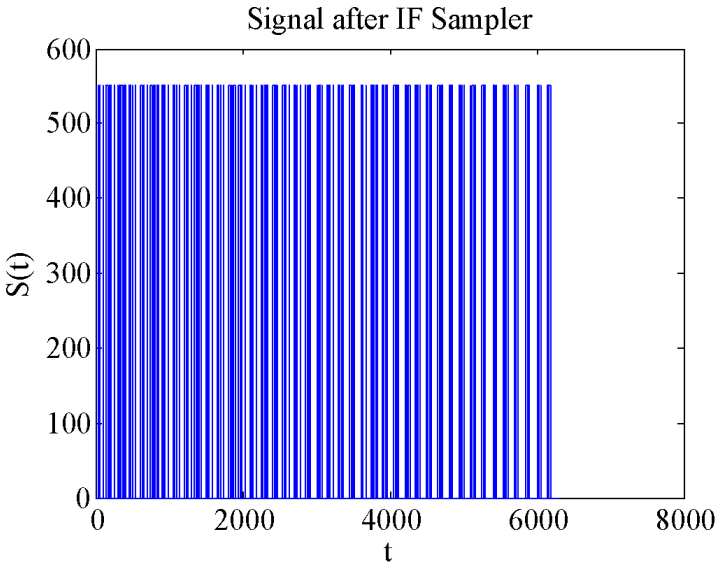


FIGURE 4.10: Square wave obtained from the LIF sampler, using the optimized RBF spike model in Fig. 4.9.

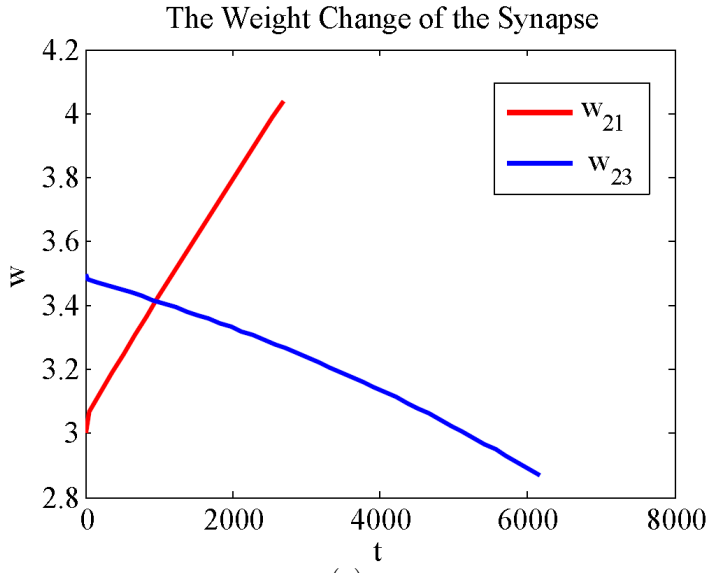
from the terrain sensors are prioritized when the virtual insect approaches rough terrain.

The coding schemes for SNNs include both rate coding, which is the frequency of neuron spikes, and temporal coding, which is correlated with the exact firing times of the neurons. This paper employs the rate coding to decode the output spikes of the neurons. The average firing rate of neuron i over time interval $[t_{l-1}, t_l]$ can be calculated by (4.35). The sensor neurons (see Fig. 3.3) receive inputs from the sensors of the virtual insect. The firing frequencies of the outputs, which connect to the motors, are depicted $f_i^p(t_l)$, where i is the index of the neuron and p is the index of the training samples. The error, δ, at time t_l is defined by (4.36).

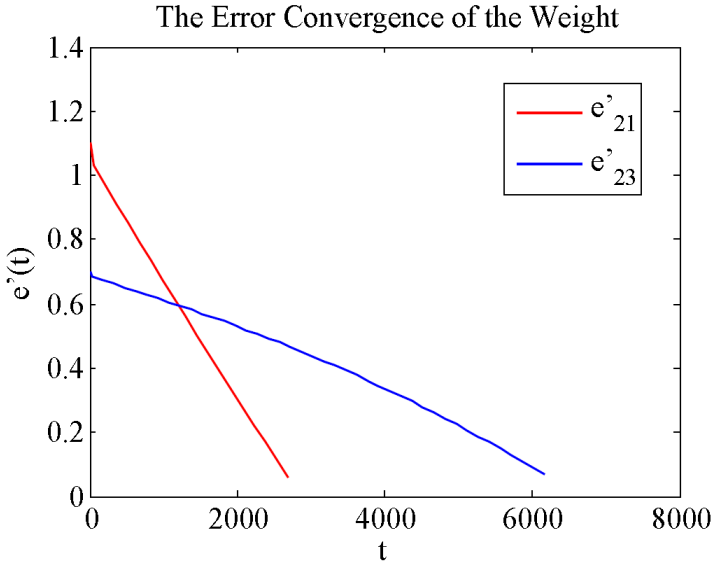
$$f_i^p(t_l) = \frac{\sum_{t_{l-1}}^{t_l} H(V_i(t_{l-1}) > V_{th})}{\Delta t}, \qquad \Delta t = t_l - t_{l-1} \tag{4.35}$$

$$\delta(t_l) = \frac{\sum_{i \in O} \sum_{p=1}^{P_m} ||F_i^p - f_i^p(t_l)||}{2 * P_m} \tag{4.36}$$

where O is the set of indices identifying the output neurons in the SNN, P_m is



(a)



(b)

FIGURE 4.11: Indirect weight changes brought about by STDP, and corresponding error functions, for the SNN in Fig. 4.8 stimulated using the input spike train in Fig. 4.10.

the total number of training samples and F_i^p is the target firing frequencies of the neuron i for training samples p, $f_i^p(t_l)$ is the firing frequency of the neuron i for training samples p during $[t_{l-1}, t_l]$.

The RBF centers are updated according to (4.37) in order to minimize the error $\delta(t)$. During each training epoch $[t_l, t_{l+1}]$, only one square pulse is injected into each neuron, and the time difference between the centers of the RBF input to neuron j and i, d_{ji}, is given by

$$d_{ji}(t) = -\lambda \frac{\partial \delta(t - \Delta t)}{\partial d_{ji}(t - \Delta t)} \tag{4.37}$$

where λ is a constant learning rate.

Then, the centers of RBF pulses to each neuron can be calculated by solving the linear systems,

$$d_{ji} = c_j - c_i, \qquad i, j \in [1, 2, \cdots, N] \tag{4.38}$$

where i, j are the index of neurons, and N is the total number of neurons in the SNN.

At every time step, d_{ji} is calculated as described above, and the values of c are calculated using (4.38). Then we input the RBF to the neurons during each epoch $[t_l, t_{l+1}]$. A square pulse with the center c is injected into each training neuron, such that the SNN is trained by our indirect training method via the STDP rule discussed in Section 2.3. Fig. 4.12 shows the optimized firing times of neurons 1 and 3 caused by the RBF inputs and the corresponding weight changes due to the STDP rule during training. If the output error is lower than δ_{min}, the training stops and the trained SNN is used on the virtual insect problem.

4.3 Indirect Training by Weight Perturbations

The output signal from the output neuron layer is decoded using rate coding, which computes the mean firing frequency of a set of neurons and has been chosen due to

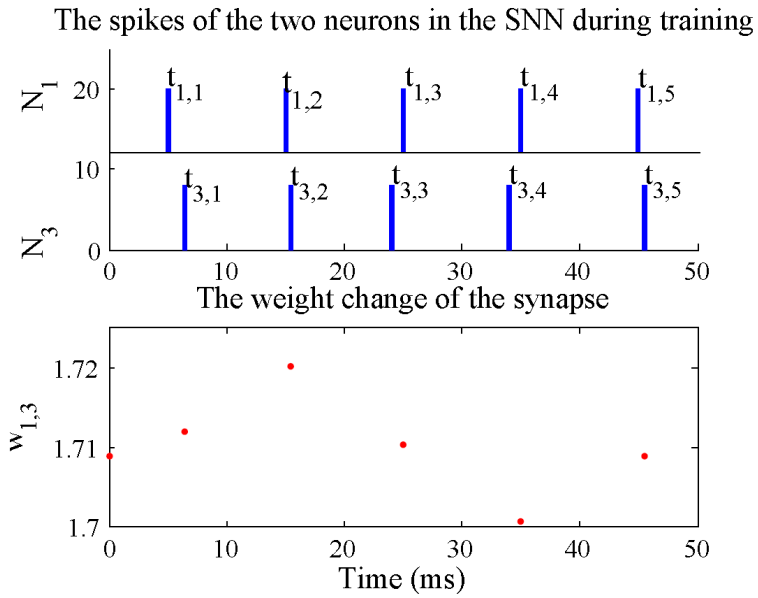


FIGURE 4.12: Optimized input spike train, and indirect weight changes brought about by STDP.

its capability to successfully describe the behavior in some biological neural systems, such as the decoding of signals from muscle stretch receptor neural cells Mountcastle (1957); Hubel and Wiesel (1959). In the rate coding method, the mean firing frequency of a set of K neurons is calculated as,

$$f(t_r) = \frac{\Sigma_{i=1,\ldots,K} z_i(t_r)}{K} \tag{4.39}$$

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 neurons, Eq. (4.39) can calculate the left motor neurons' mean firing frequency. If K are the right motor neurons, Eqn. (4.39) can calculate the right motor neurons' mean firing frequency.

The SNN can be trained to control the virtual insect to accomplish the objectives of the navigation problem described in Sec. 3. The training is achieved through the control of precise training stimuli given to the network which, in turn, indirectly modifies the SNN 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

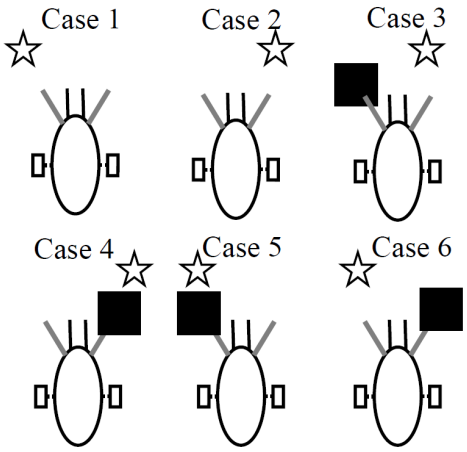


FIGURE 4.13: Insect locations for six training cases.

form of $m = 6$ cases of sensor stimuli in Fig. 3.6 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. (4.40).

Same as the desired output for gradient training method, the desired output signal of the SNN can be defined as,

$$\begin{pmatrix} f_L^* \\ f_R^* \end{pmatrix} = \begin{pmatrix} h_L & g_L \\ h_R & g_R \end{pmatrix} \begin{pmatrix} \kappa \\ \eta \end{pmatrix} \tag{4.40}$$

where f_L^* and f_R^* are the desired average firing frequencies of the left and right motor neurons, and h_L, g_L, h_R, and g_R are the sensor input stimuli from Eq. (3.4) and Eq. (3.5). The constants κ and η are constants that are chosen such that $\kappa > \eta$ to prioritize obstacle avoidance over reaching the target.

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. 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. (3.4) and Eq. (3.5). 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,

$$e_{k,\ell} = \frac{\sqrt{\sum_{i=1,...,m}[\mathbf{u}_i^* - \mathbf{u}_i]^T[\mathbf{u}_i^* - \mathbf{u}_i]}}{m} \tag{4.41}$$

where $\mathbf{u}_i = [f_R\ f_L]^T$ is the decoded control output of the SNN for training case i, $\mathbf{u}^* = [f_R^*\ f_L^*]^T$ is the desired output from Eq. (4.40), and ℓ is an index corresponding to the two testing phases per epoch.

Fig. 4.14 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 an experimental training signal 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. (4.39) with the desired output computed from Eq. (4.40). For each training case, the sensory stimuli is encoded as constant current injections, defined by Eq. (3.4) and Eq. (3.5), 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

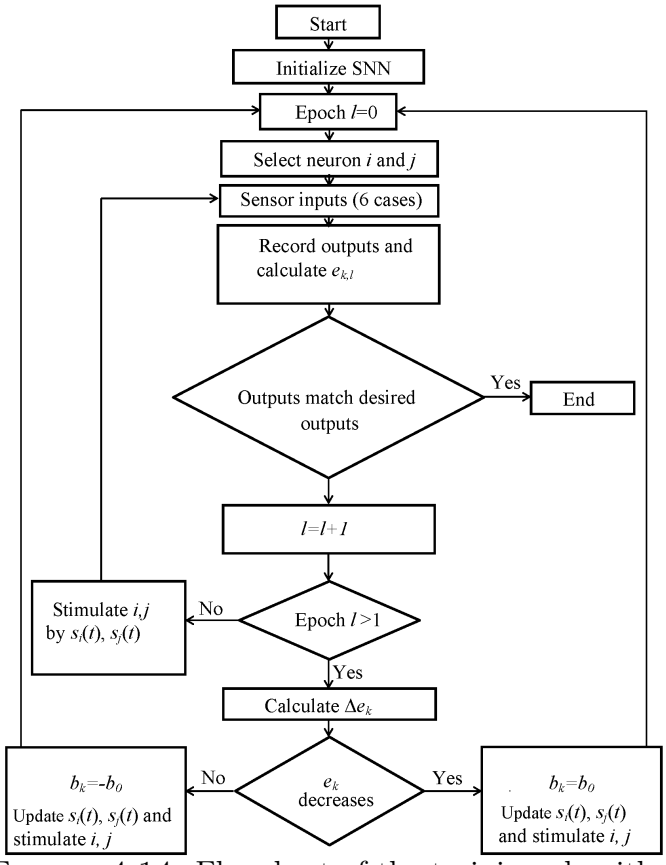


FIGURE 4.14: Flowchart of the training algorithm.

and be reliably decoded as an output. Then $e_{k,1}$ is obtained from Eq. (4.41).

The *initial training phase* of the epoch involves the delivery of an experimental 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 experimental 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,

$$s_i(t) = w \sum_{a=1}^{G} \left[H\left(t - p_{i,a} + \frac{\beta}{2}\right) - H\left(t - p_{i,a} - \frac{\beta}{2}\right) \right] \tag{4.42}$$

where $p_{i,a}$ represents the temporal center of the a^{th} square pulse delivered to the i^{th} input neuron, $G = 10$ is the number of square pulses in the 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. 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. (4.41) 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. (4.42), 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,

$$b_k = -\operatorname{sgn}(\Delta e_k) b_0 \tag{4.43}$$

Then the square pulses are 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 **?**, the optimal number of square pulses, G^*, is a function of b_k and Δe_k, and is given by,

$$G^* = \begin{cases} 2G, & b_0 > 0 \text{ and } \Delta e_k > 0 \\ \frac{1}{2}G, & b_0 < 0 \text{ and } \Delta e_k < 0 \\ G, & \Delta e_k < 0 \end{cases} \tag{4.44}$$

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. (4.43), and the number of square pulses, G^* from Eq. (4.44). 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 $G^* = 10$. In Fig. 4.15, $w_{1,19}$ increases ten times according to the STDP rule. While if $\Delta e_k > 0,\ \ b_0 > 0$, the number of training input pairs is $2G = 20$ in Fig. 4.15 bottom.

In Fig. 4.15, 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.

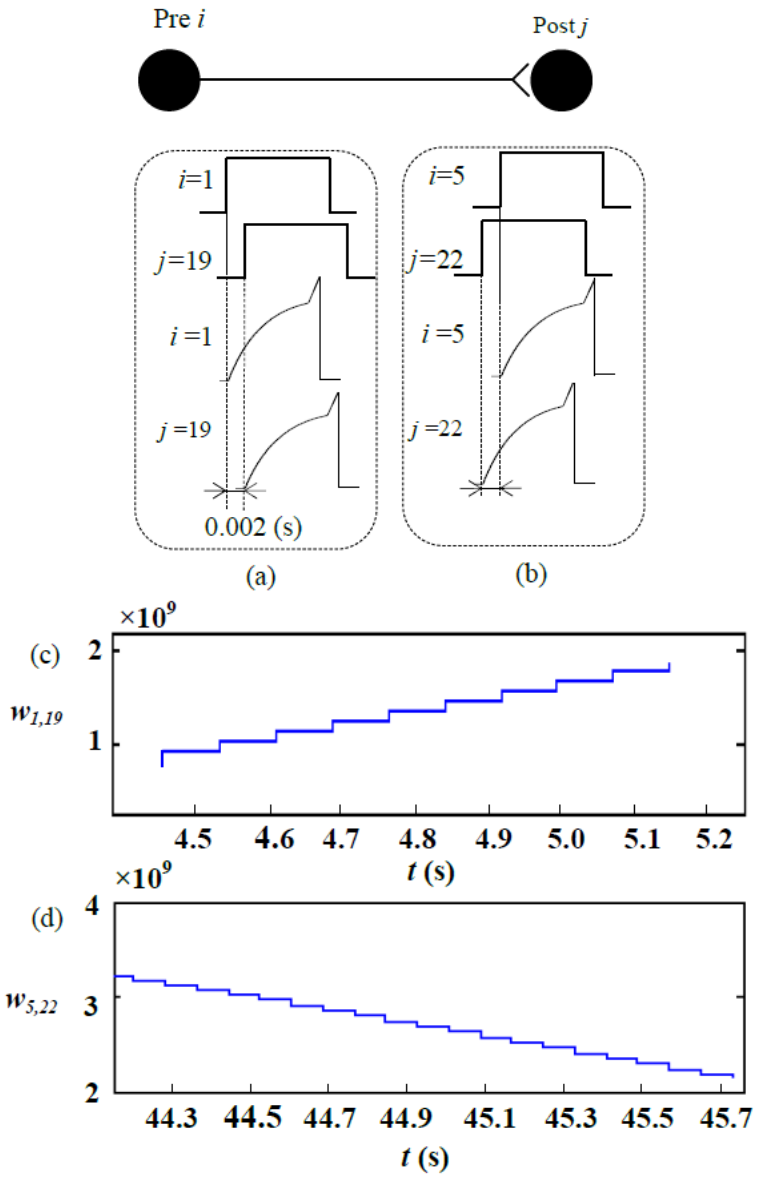


FIGURE 4.15: Action potentials of the pre and post-synaptic neurons and the weights change by training inputs.

5

Simulation Results

5.1 Seven Neurons Controlled Virtual Insect

The architecture and algorithm of the indirect training approach presented in Section 4.2 are used to train an SNN with randomized initialization, in order to perform target detection and terrain navigation. The objective of the simulations presented is to test the effectiveness of this training approach by comparing two trained states of the SNN including naive, partially-trained and fully trained on blank, s-maze, and cloud terrains.

Fig. 5.1 shows the error defined by (4.35) and (4.36) against the training duration where the error converges. Therefore, the training would be stopped once the error is lower than δ_{min}. Due to the instability of the solution caused by continuous strengthening/weakening of synapses, the synaptic weights are fixed after the training process completes. Fig. 5.2 demonstrates the evolution of synaptic weights with training inputs. The strengthened connections between terrain sensory neurons and neuron 3 (see Fig. 3.3) ensure the priority of terrain information; meanwhile the weights of inhibitory synapses are updated so that the both sides of SNN structure can be bal-

anced. The synapses connecting with motor neurons can either be strengthened or weakened as long as the values of these synaptic weights are comparable to balance the two motor outputs.

The simulations are conducted in MATLAB® and in order to create the virtual environment, a 600×600 pixels image of the terrain and the target were generated. The initial positions of the virtual insect differ in the three environments. As illustrated by the examples in Section 5.1.1 to 5.2.2, the indirect training method is capable of both strengthening and weakening synapses without directly manipulating synaptic weights. It is also capable of integrating information regarding the target location and terrain conditions, and, thus, it can train the virtual insect to avoid rough terrain on its path to the target.The movie clips for these results show a very realistic insect behavior and can be downloaded from the URL at Zhang and Xu (2013).

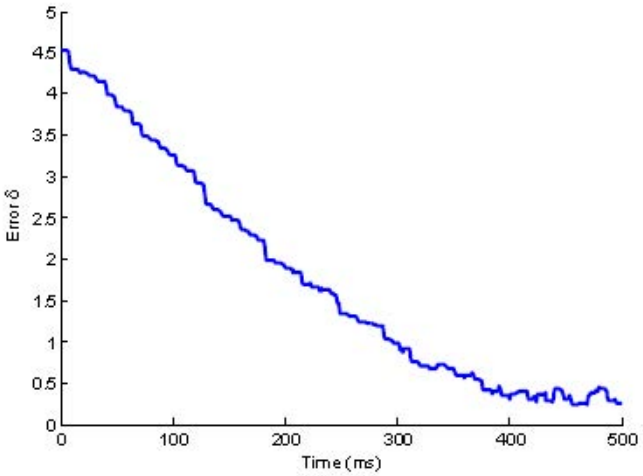


FIGURE 5.1: Error δ during training of the neural network with 7 neurons.

5.1.1 Blank Terrain

The properties and effectiveness of the indirect training are first tested in a simple environment where the terrain has uniform smoothness and, therefore, only target

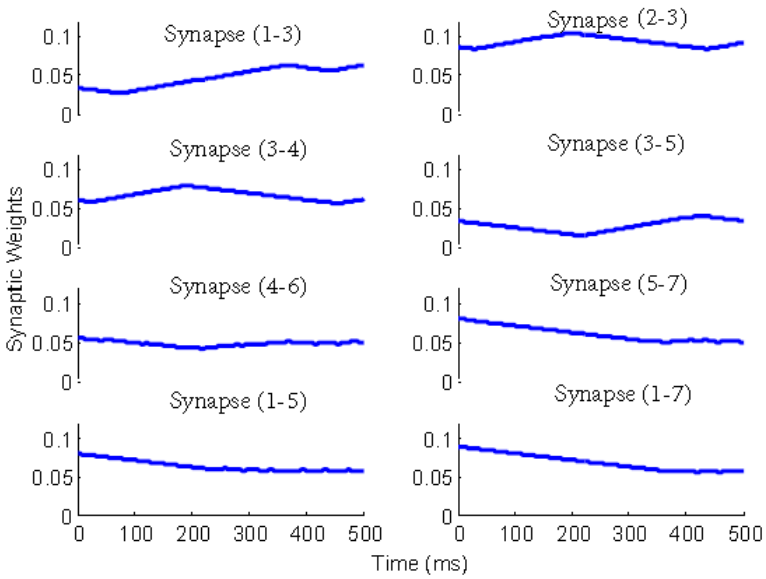


FIGURE 5.2: Evolution of eight synaptic weights subject to STDP during training.The synapse between neuron 1 and 3 is labeled as synapse (1-3). See Fig. 3.3 for all connections.

information is relevant. As shown by the trajectory in Fig. 5.3-5.16, initially, the virtual insect rotates randomly in place. After partially trained, the virtual insect moves through the workspace but does not approach the target. Finally, following the completion of the training procedure, the virtual insect approaches the target using the path of shortest distance.

5.1.2 S-maze Terrain

In this scenario, the virtual insect must not only find the target, but integrate information about the terrain. The simulation results for naive, partially trained and fully trained states are illustrated in Fig. 5.5-5.6. The naive insect rotates without moving toward the target. In the partially-trained state, the insect initially moves away from the target and due to its capacity of terrain navigation, it successfully accomplishes the task by rambling along the black terrain.

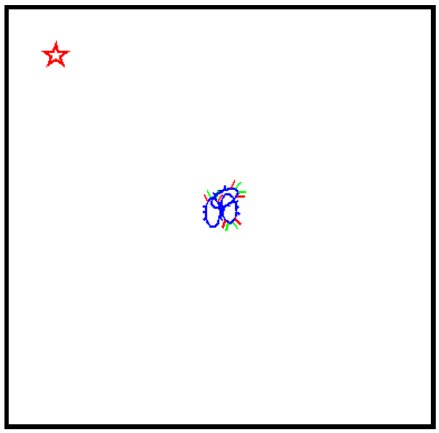


FIGURE 5.3: Trace of naive insect controlled by 7 neurons on blank terrain.

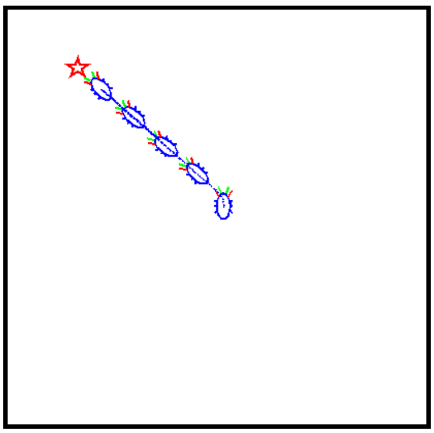


FIGURE 5.4: Trace of fully-trained insect controlled by 7 neurons on blank terrain .

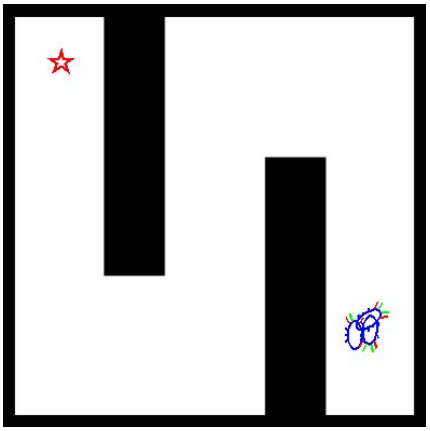


FIGURE 5.5: Trace of the naive insect controlled by 7 neurons on an S-maze terrain (see movie in Zhang and Xu (2013)).

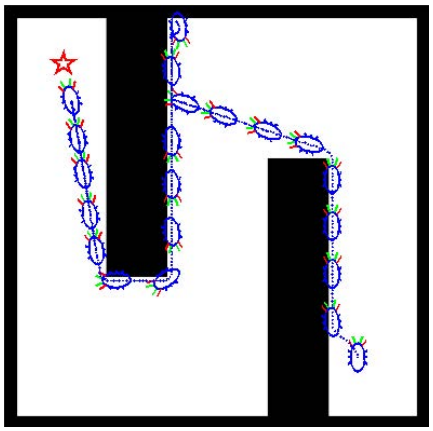


FIGURE 5.6: Trace of the fully-trained insect controlled by 7 neurons on an s-maze terrain (see Zhang and Xu (2013)).

5.1.3 Cloud Terrain

The cloud terrain is a heavily obstacle populated maze, created via Photoshop®. This environment introduces rough terrain and narrow channels, creating a complex and difficult landscape for the virtual insect to navigate. In the partially-trained case, the insect fails to acquire the target as it does in the s-maze terrain because of the complexity of the cloud terrain. As expected, the fully trained SNN accounts for

both target location and terrain roughness and effectively controls the virtual insect along its path.

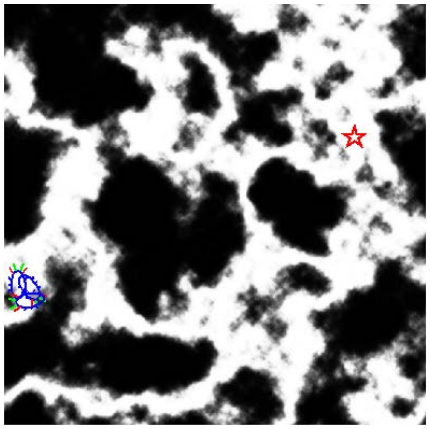


FIGURE 5.7: Trace of the naive insect controlled by 7 neurons on the cloud terrain (see movie in Zhang and Xu (2013)).

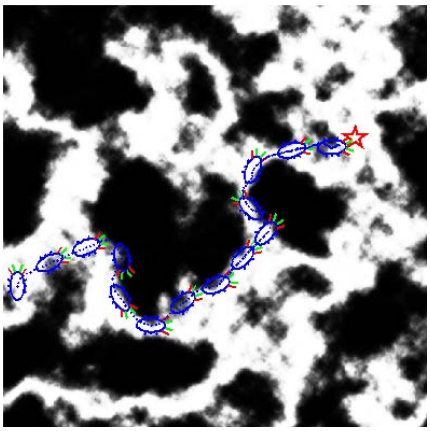


FIGURE 5.8: Trace of the fully-trained insect controlled by 7 neurons on cloud terrain (see movie in Zhang and Xu (2013)).

5.2 Insect Trace Controlled by Larger Neural Networks Trained by Weight Perturbations

In Fig. 5.9 (a), the sensor input g_L stimulates the left target sensor neurons, which are the left bottom of the input layer. Fig. 5.9 (b) shows the activity of the three layers during the stimulation of the left bottom neurons. Before training, the left motor neurons do not fire. Fig. 5.9 (c) shows the membrane potentials just after stopping the sensor inputs. The membrane potentials are still above the resting potential 0.014. During the training in Fig. 5.9 (d), two neurons in the input layer are stimulated for training. After training, in Fig. 5.9(e), the left motor neurons fire in order to force the insect to turn right.

Fig. 5.10 (a) shows that the error defined in Eq. (4.41) converges, which cannot be further improved with more training. Therefore, the training input would be removed once the error is lower than a threshold value. By comparing Fig. 5.10 (b) and (c) in

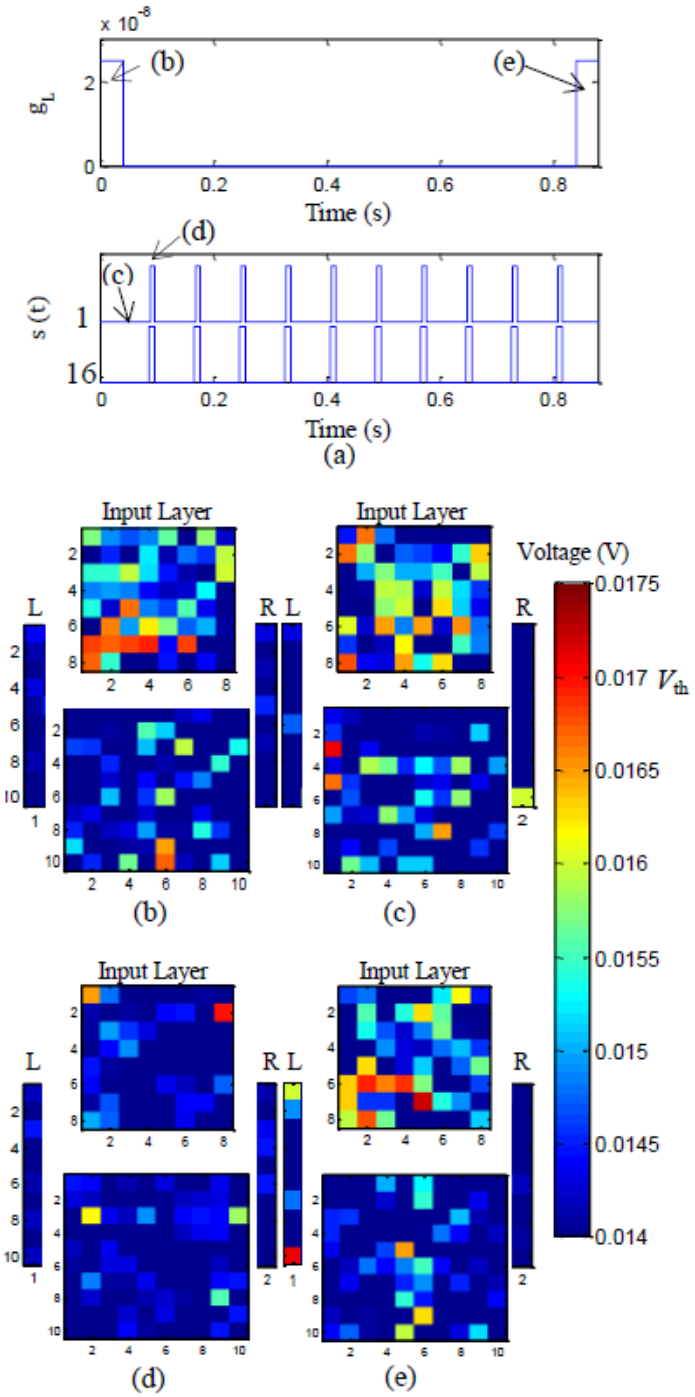


FIGURE 5.9: Membrane potentials during tests for case 2 (no obstacle and target right). (a). Test and training inputs injected to the neurons. (b - e). Membrane potentials of the neurons in four different time instants marked in (a).

case 2 (no obstacle right target), the left motor neurons fire more than the right motor neurons in order to force the insect to turn right after training. The properties and

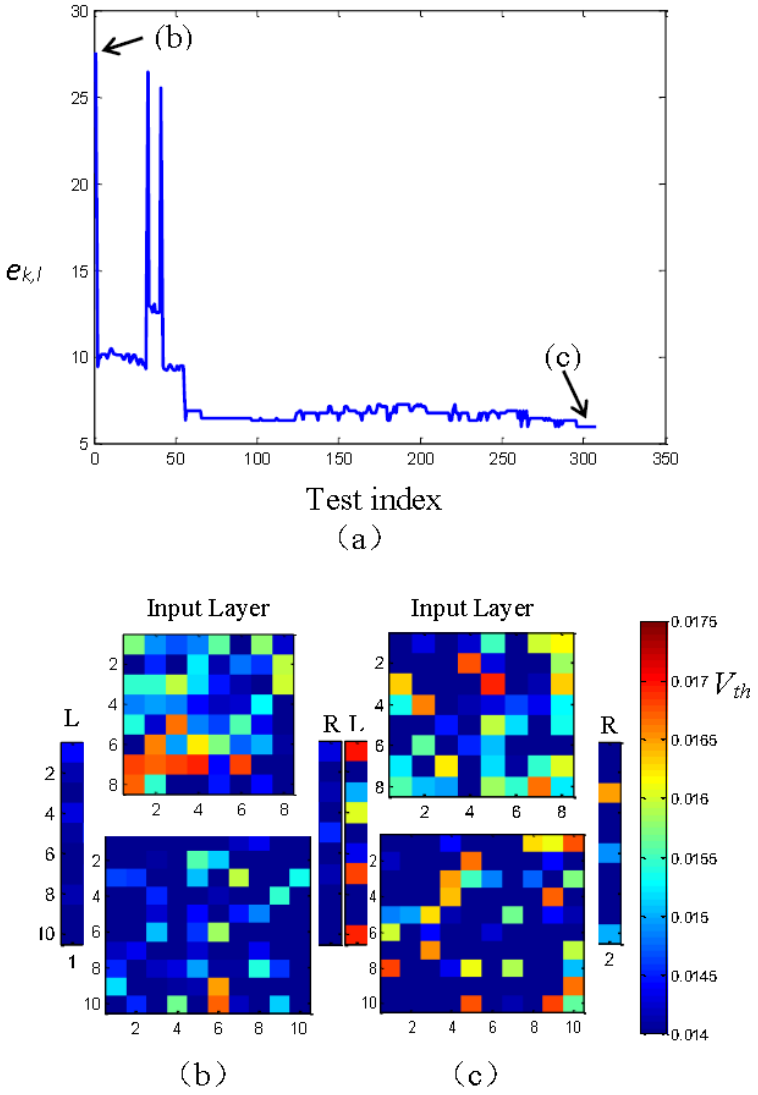


FIGURE 5.10: Membrane potential of three layers before the training and after training. (a). Error change during the entire simulation. (b-c). Membrane potentials of the neurons in three layers at the times pointed in (a).

effectiveness of the indirect training are first tested in a simple environment where the terrain has uniform smoothness and, therefore, only target information is relevant. As shown by the trajectory in Fig. 5.15-5.16, initially, the virtual insect randomly

rotates in place. Following the completion of the training procedure, the virtual insect approaches the target using the path of shortest distance.

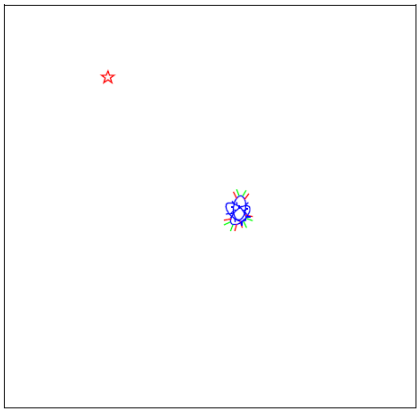


FIGURE 5.11: Naive insect on a blank terrain controlled by the neural network with 184 neurons.

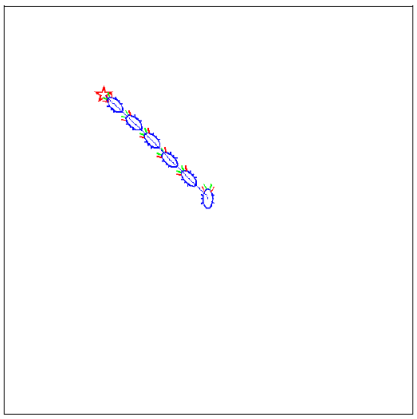


FIGURE 5.12: Trained insect on a blank controlled by the neural network with 184 neurons.

5.2.1 S-maze

In this scenario, the virtual insect must not only find the target, but also integrate information about the terrain. The simulation results for naive and fully trained states are illustrated in Fig. 5.13-5.14. The naive insect rotates and moves away from the target. After training, the trained virtual insect can navigate in the S-maze and reach the target.

5.2.2 Cloud Terrain

In Fig. 5.7 and Fig. 5.8, the cloud terrain is a heavily obstacle populated maze, created via Photoshop®. This environment introduces rough terrain and narrow channels, creating a complex and difficult landscape for the virtual insect to navigate. The naive insect can avoid some obstacles but it fails to detects the target. As expected, the trained SNN accounts for both target location and terrain steepness and effectively controls the virtual insect along its path.

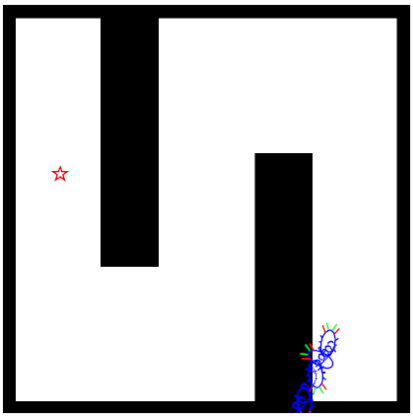


FIGURE 5.13: Naive insect on an S-Maze controlled by the neural network with 184 neurons.

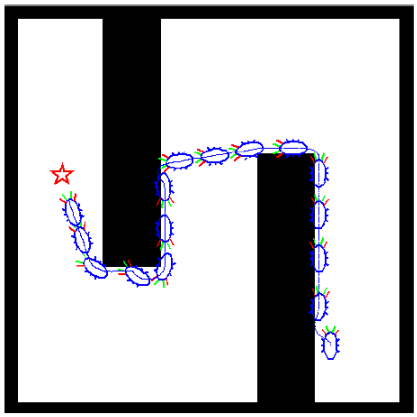


FIGURE 5.14: Trained insect on an S-Maze controlled by the neural network with 184 neurons.

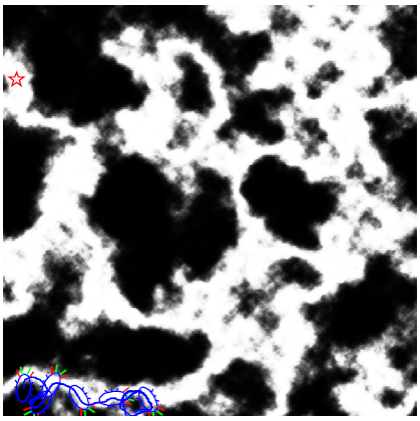


FIGURE 5.15: Naive insect on a Cloud terrain controlled by the neural network with 184 neurons.

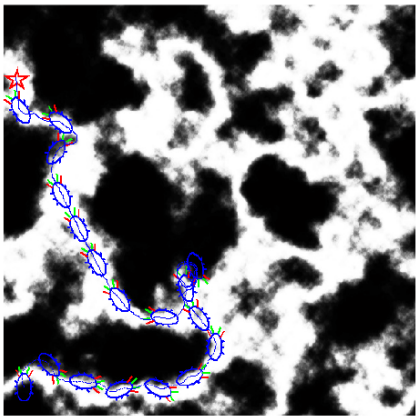


FIGURE 5.16: Trained insect on a Cloud terrain of controlled by the neural network with 184 neurons.

5.3 Tripod Insect Controlled by a Large SNN

The architecture and algorithm of the indirect training approach presented in Section 4.3 are used to train a three-layer SNN (550 neurons) with recurrent connections, for the control of the virtual insect in the navigation problem described in Section 3.2. Fig. 5.17 shows that the error defined in (4.41) converges, which cannot be further improved with more training. Therefore, the training input is removed once the error is less than a tolerance δ_{min}. The objective of the simulations presented

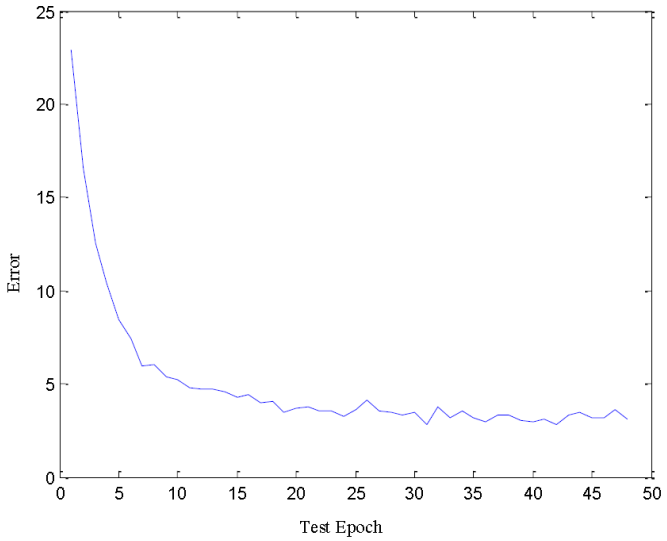


FIGURE 5.17: Error during training of the SNN with 550 neurons.

is to evaluate the effectiveness of this training approach by comparing two states of the SNN, naive and fully trained, on identical terrains.

The terrain used for the simulations is the cloud terrain. The naive insect can travel to the target, but it cannot avoid the obstacles and navigate the maze. As illustrated by Fig. 5.18 and Fig. 5.19, the indirectly trained SNN is capable of integrating information regarding the target location and terrain conditions, and, thus, the indirect training approach can train the virtual insect to avoid rough terrain on its path to the target.

The fluctuations of membrane potentials of SNN during testing are demonstrated in Fig. 5.20. In Fig. 5.20, only the right bottom area receives sensory inputs, which propagate to the other two layers as shown in Fig. 5.21.

5.4 Strength of Large Size SNNs

Without sensor noise, there is not much difference between the behavior of small neural network controlled virtual insect and large neural network controlled virtual

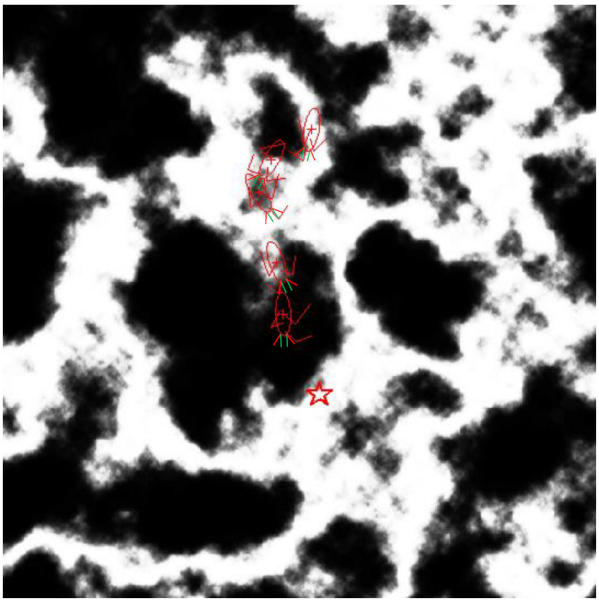


FIGURE 5.18: Trace of the untrained virtual insect controlled by 550 neurons on cloud terrain.

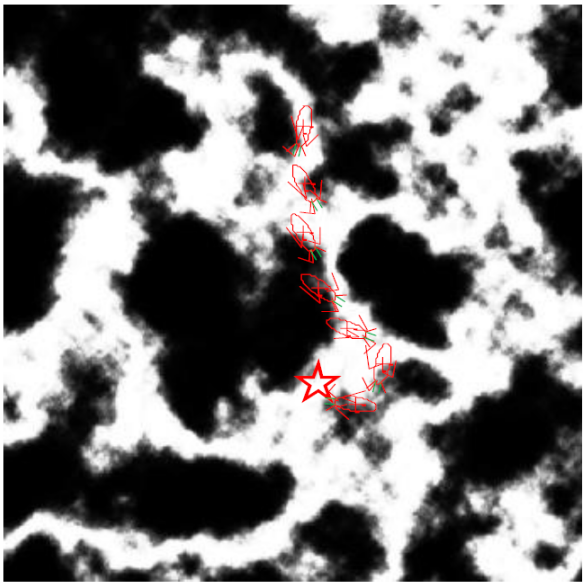


FIGURE 5.19: Trace of the fully-trained virtual insect controlled by 550 neurons on cloud terrain.

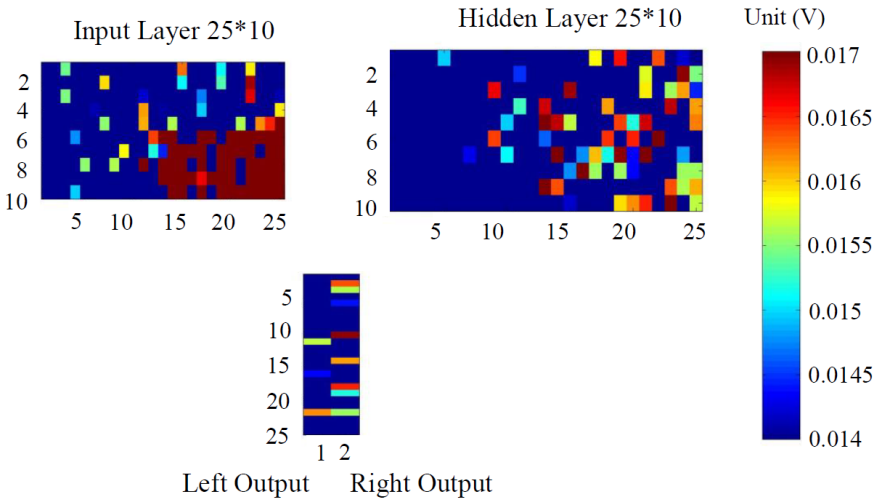


FIGURE 5.20: Distribution of membrane potentials when the neurons in the right bottom receive sensory inputs.

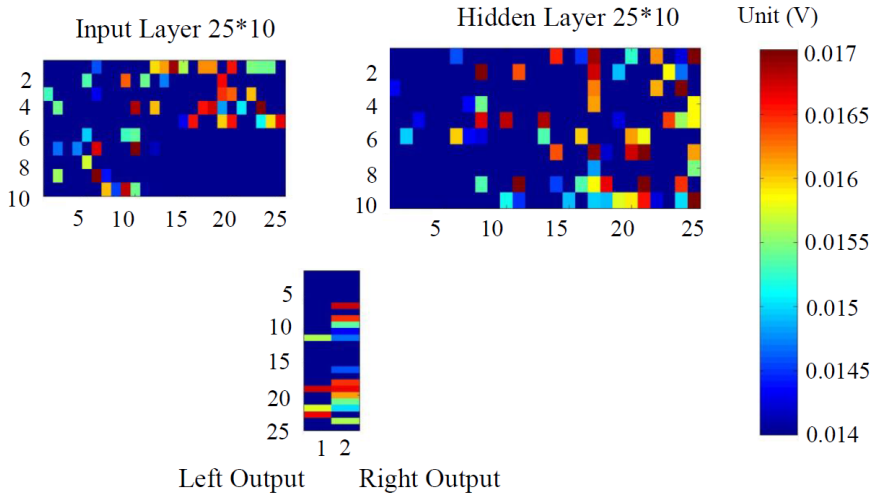


FIGURE 5.21: Propagation of sensor inputs to other layers of the SNN.

insect. In order to find whether the larger neural network has some strengths than the small network, some sensor noises are added to test whether the behavior of larger neural network is better than the small neural network.

In order to compare the performance of the two different size neural networks, a noise generated by a normal distribution is added to the four sensor inputs. For comparing, the noise is chosen to be 50 percent of the original sensor values. Therefore, the sensor value with noise is,

$$S_i = 0.5 * rand + S_i^0 \tag{5.1}$$

where i is the index of the four sensors, $rand$ is a random number following the normal distribution between -1 and 1, S_i^0 is the sensor values without noise.

First, we test the 7 nodes neural network given the sensor with noise. The insect is located in the middle of a 600*600 blank terrain. The target is on the left front of the insect.

We can see that the insect can not reach the target if there is a large sensor noise. However, the figure below shows the behavior of the 3 layer neural network with 184 neurons. After added noise defined in Eqn. 5.1, this large neural network can still reach its target.

Therefore, in conclusion, the neural network with 184 neurons are much more adaptive to the sensor noises than the neural network with 7 neurons.

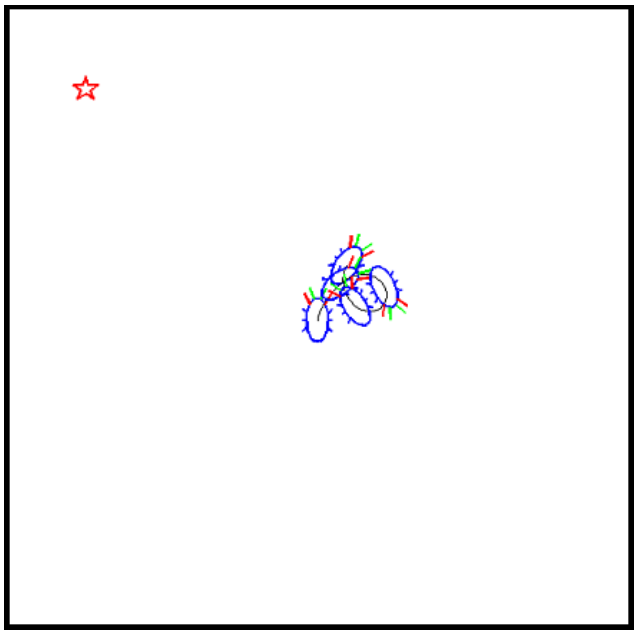


FIGURE 5.22: Trained insect controlled by 7 neurons with sensor noise on a blank terrain.

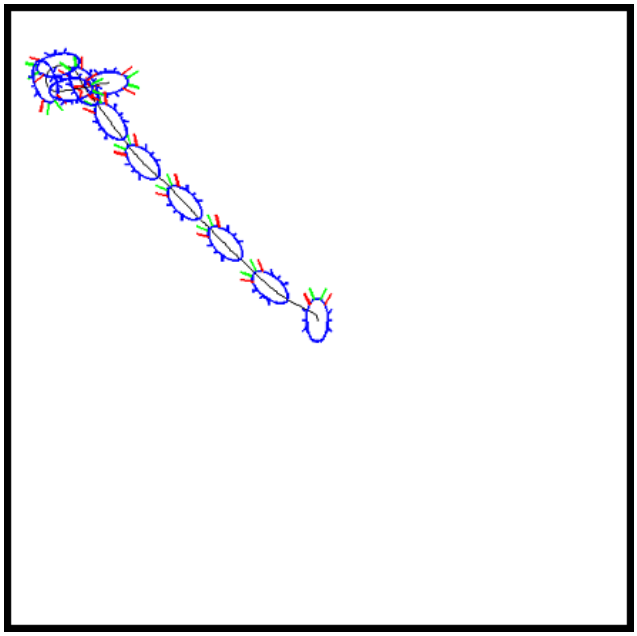


FIGURE 5.23: Trained insect controlled by 184 neurons with sensor noise on a blank terrain.

6

Conclusion

Because our training methods are biologically plausible, they will be applied to CMOS/memristor neuromorphic computers, that do not allow the direct manipulations of the synaptic weights, that are instead required by existing SNN training algorithms. The trained neuromorphic chips cost low energy with effective computational power. Even though some papers have published training methods on training a simulated neuromorphic system by directly setting the connection strengths, these methods are not applicable to neuromorphic chips in hardware. By indirect training algorithms, a trained neuromorphic chip will be used to solve many control problems, such as robot path planning, cognitive computing and the human brain project. The future work aims to develop methods for porting higher level algorithms, specifically Neurodynamic programming (NDP), from a purely stand alone software implementation on the microprocessor to an Application Specific Integrated Circuit (ASIC) design. NDP is a general approach for designing control algorithms for nonlinear, non-convex dynamical systems that learn and adapt in real time, subject to new and unforeseen environments. The optimization of this adaptability in control with respect to a changing environment is vital to the development and maturation of

Cyber-Physical Systems (CPS). A control system ASIC will vertically integrate embedded software computing, energy-optimized hardware, effective communication between modules on the ASIC, and the control algorithms used for the specified control. The ASIC is envisioned to use CMOS/memristor hardware in order to adopt the advantages of neuromorphic computing. CMOS/memristor-devices have the potential for creation of adaptive, neuromorphic sensorimotor circuits for intelligent robots and neuroprosthetic devices that can adaptively interact in uncertain environments. The use of CMOS/memristors will enable an approximated synaptic plasticity, device density, scalability, and fault tolerance inherent in biological neural networks. One objective of the future research will involve the study and development of CMOS/memristor hardware for embedded CPS applications. The ultimate objective is not just a virtual realization of the CMOS/memristor hardware but to solve issues related to CMOS and memristor integration and provide results relating to the effectiveness of such a computational platform.

Another future project will develop and verify a mathematical and physiological framework for uncovering the system-level capabilities of mammalian brains that enable them to optimally solve complex control and system identification problems, by adapting to uncertain and non-convex environments. Although existing designs are very effective at controlling systems whose dynamics are approximated a priori, they are not yet capable of handling the range of operating conditions, unforeseen damages, and failures handled by biological brains. In principle, neurodynamic programming (NDP) can solve non-convex optimal control problems adaptively. But, the current formalisms for artificial neural networks (ANNs) and gradient-based learning are far removed from their biological counterparts. By integrating control theory with in silico and in vitro experiments, this project will develop NDP algorithms that are biologically plausible and testable on light-sensitive neurons grown in culture. The cultured neurons will be trained using light patterns, and their spatiotem-

poral responses will be measured in real time using microelectrode arrays (MEAs). Furthermore, it will be possible to introduce lesions and to observe how memory is restored, either spontaneously by outgrowth of existing neurons, or by re-seeding the cultures. This project will aim to reverse-engineer dopamine and cortical neuronal cultures on a chip, emulating the actor and critic networks, respectively, by training them to perform adaptive control of aircraft through an NDP architecture. The outcome of this research shall help uncover the mechanisms by which sensorimotor learning is influenced by dopamine neuron activity in the basal ganglia, which is believed to function as a reinforcement signal.

It has recently been demonstrated that the higher control of insect movements in highly conserved region of insect brain call the Central Complex (CX) Pfeiffer and Homberg (2014); Strausfeld (2012); Strausfeld and Hirth (2013); Strauss (2002). The future research is to train the CX of a behaving cockroach, by implementing a novel spike-based perturbative approach in-vivo. The approach relies on a learning rule for manipulating patterns of neural stimulations (spike trains) that can be delivered using a four-tetrode arrangement inserted into the cockroach CX, modulating synaptic plasticities based on the observed locomotory behaviors. The neural system of the virtual insect will change to a more biologically plausible one with different regions perform different functions.

As part of the insect navigation research, the existing CPG and locomotion models will be improved by considering the three-dimensional joint kinematics and neurobiology. Then, the improved CPG network and locomotion model will be connected to the CX model, in order to obtain a more realistic insect simulation.

Future study will also focus on implementing our training algorithms on flying micro-robots, which have more complex dynamics. Because micro-robots need to have a control system that is not only effective but also light enough, the Neuromorphic chip can be a good choice because of its less energy cost and effectiveness in

control tasks. First of all, our indirect training algorithms will be tested on training the neuro-morphic chip to control a flying micro-robot for hovering like qudrators or bees. The neural network controlled micro-robot will be more adaptive to some emergent cases and unknown environments than traditional controllers. After the test on hovering, the training methods will be tested on flying insects like RoboBees, controlled by sensor-motor nervous systems for searching, rescuing and hazardous environment exploration.

Appendix A

Appendix

$$\frac{\partial E^2}{\partial c_1} = \begin{cases} 2E[\frac{g_+(t_{1,1},t_{1,3})}{\tau_+(1+g_+(t_{1,1},t_{1,3})}] & \text{if } v(t_{1,3}+\tau_d,1,3) > V_{th} \\ 2E[\frac{g_+(t_{1,1},t_{1,2})}{\tau_+(1+g_+(t_{1,1},t_{1,2}))}] & \text{if } v(t_{1,2}+\tau_d,1,2) > V_{th} \\ 2E[\frac{g_+(t_{1,1},t_{1,4})}{\tau_+(1+g_+(t_{1,1},t_{1,4}))}] & \text{if } v(t_{1,4}+\tau_d,1,4) > V_{th} \\ 2E[\frac{g_+(t_{1,1},t_{1,5})}{\tau_+(1+g_+(t_{1,1},t_{1,5}))}] & \text{if } v(t_{1,5}+\tau_d,1,5) > V_{th} \end{cases} \tag{A.1}$$

$$\frac{\partial E^2}{\partial c_2} = \begin{cases} 2E[\frac{g_+(t_{1,2},t_{1,3})}{\tau_+(1+g_+(t_{1,2},t_{1,3})}] & \text{if } v(t_{1,3}+\tau_d,1,3) > V_{th} \\ 2E[\frac{-g_+(t_{1,1},t_{1,2})}{\tau_+(1+g_+(t_{1,1},t_{1,2}))}] & \text{if } v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,5}+\tau_d,3,5) > V_{th}, \\ & c_3 > \frac{c_5+c_2+2\tau_d}{2} \\ 2E[\frac{-g_+(t_{1,1},t_{1,2})}{\tau_+(1+g_+(t_{1,1},t_{1,2}))} + \frac{g_-(t_{1,2},t_{1,3})}{\tau_-(1+g_-(t_{1,2},t_{1,3}))}] & \text{if } v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,4}+\tau_d,3,4) > V_{th}, \\ & c_3 < \frac{c_4+c_2+2\tau_d}{2} \text{ or} \\ & v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,5}+\tau_d,3,5) > V_{th}, \\ & c_3 < \frac{c_5+c_2+2\tau_d}{2} \\ 2E[\frac{-g_+(t_{1,1},t_{1,2})}{\tau_+(1+g_+(t_{1,1},t_{1,2}))} + \frac{g_-(t_{1,2},t_{1,3})}{\tau_-(1+g_-(t_{1,2},t_{1,3}))} & \\ +\frac{g_-(t_{1,2},t_{1,4})}{\tau_-(1+g_-(t_{1,2},t_{1,4}))} + \frac{g_-(t_{1,2},t_{1,5})}{\tau_-(1+g_-(t_{1,2},t_{1,5}))}] & \text{if } v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,5}+\tau_d,3,5) < V_{th} \\ 2E[\frac{g_+(t_{1,2},t_{1,4})}{\tau_+(1+g_+(t_{1,2},t_{1,4})}] & \text{if } v(t_{1,4}+\tau_d,1,4) > V_{th} \end{cases} \tag{A.2}$$

$$\frac{\partial E^2}{\partial c_3} = \begin{cases} 2E\big[\frac{-g_+(t_{1,1},t_{1,3})}{\tau_+(1+g_+(t_{1,1},t_{1,3})} + \frac{-g_+(t_{1,2},t_{1,3})}{\tau_+(1+g_+(t_{1,2},t_{1,3})} \\ +\frac{g_-(t_{1,3},t_{1,4})}{\tau_-(1+g_-(t_{1,3},t_{1,4})}\big] & \text{if } v(t_{1,3}+\tau_d,1,3) > V_{th}, \\ & v(t_{1,5}+\tau_d,4,5) > V_{th}, \\ & c_3 > 2c_4 - c_5 - 2\tau_d \\ 2E\big[\frac{-g_+(t_{1,1},t_{1,3})}{\tau_+(1+g_+(t_{1,1},t_{1,3})} + \frac{-g_+(t_{1,2},t_{1,3})}{\tau_+(1+g_+(t_{1,2},t_{1,3})}\big] & \text{if } v(t_{1,3}+\tau_d,1,3) > V_{th}, \\ & v(t_{1,5}+\tau_d,4,5) > V_{th}, \\ & c_3 < 2c_4 - c_5 - 2\tau_d \\ 2E\big[\frac{g_+(t_{1,3},t_{1,5})}{\tau_+(1+g_+(t_{1,3},t_{1,5}))}\big] & \text{if } v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,5}+\tau_d,3,5) > V_{th}, \\ & c_3 > \frac{c_5+c_2+2\tau_d}{2} \text{ or} \\ & v(t_{1,5}+\tau_d,1,5) > V_{th} \\ 2E\big[\frac{-g_-(t_{1,2},t_{1,3})}{\tau_-(1+g_-(t_{1,2},t_{1,3}))}\big] & \text{if } v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,5}+\tau_d,3,5) > V_{th}, \\ & c_3 < \frac{c_5+c_2+2\tau_d}{2} \text{ or} \\ & v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,4}+\tau_d,3,4) > V_{th}, \\ & c_3 < \frac{c_4+c_2+2\tau_d}{2} \text{ or} \\ & v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,5}+\tau_d,3,5) < V_{th} \\ 2E\big[\frac{g_+(t_{1,3},t_{1,4})}{\tau_+(1+g_+(t_{1,3},t_{1,4})}\big] & \text{if } v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,4}+\tau_d,3,4) > V_{th}, \\ & c_3 > \frac{c_4+c_2+2\tau_d}{2} \text{ or} \\ & v(t_{1,4}+\tau_d,1,4) > V_{th} \\ 2E\big[\frac{-g_+(t_{1,1},t_{1,3})}{\tau_+(1+g_+(t_{1,1},t_{1,3})} + \frac{-g_+(t_{1,2},t_{1,3})}{\tau_+(1+g_+(t_{1,2},t_{1,3})} + \\ \frac{g_-(t_{1,3},t_{1,4})}{\tau_-(1+g_-(t_{1,3},t_{1,4})} + \frac{g_-(t_{1,3},t_{1,5})}{\tau_-(1+g_-(t_{1,3},t_{1,5})}\big] & \text{if } v(t_{1,3}+\tau_d,1,3) > V_{th}, \\ & v(t_{1,5}+\tau_d,4,5) < V_{th} \end{cases} \tag{A.3}$$

$$\frac{\partial E^2}{\partial c_4} = \begin{cases} 2E\left[\frac{-g_-(t_{1,3},t_{1,4})}{\tau_-(1+g_-(t_{1,3},t_{1,4})}\right] & \text{if } v(t_{1,3}+\tau_d,1,3) > V_{th}, \\ & v(t_{1,5}+\tau_d,4,5) > V_{th}, \\ & c_4 < \frac{c_3+c_5+2\tau_d}{2} \text{ or} \\ & v(t_{1,3}+\tau_d,1,3) > V_{th}, \\ & v(t_{1,5}+\tau_d,4,5) < V_{th} \\ 2E\left[\frac{g_+(t_{1,4},t_{1,5})}{\tau_+(1+g_+(t_{1,4},t_{1,5})}\right] & \text{if } v(t_{1,3}+\tau_d,1,3) > V_{th}, \\ & v(t_{1,5}+\tau_d,4,5) > V_{th}, \\ & c_4 > \frac{c_3+c_5+2\tau_d}{2} \text{ or} \\ & v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,5}+\tau_d,3,5) > V_{th}, \\ & c_3 > \frac{c_2+c_5+2\tau_d}{2} \text{ or} \\ & v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,5}+\tau_d,3,5) > V_{th}, \\ & c_3 < \frac{c_2+c_5+2\tau_d}{2} \\ 2E\left[\frac{-g_+(t_{1,3},t_{1,4})}{\tau_+(1+g_+(t_{1,3},t_{1,4})} + \frac{g_-(t_{1,4},t_{1,5})}{\tau_-(1+g_-(t_{1,4},t_{1,5})}\right] & \text{if } v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,4}+\tau_d,3,4) > V_{th}, \\ & c_4 < 2c_3 - c_2 - 2\tau_d \\ 2E\left[\frac{g_-(t_{1,4},t_{1,5})}{\tau_-(1+g_-(t_{1,4},t_{1,5})}\right] & \text{if } v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,4}+\tau_d,3,4) > V_{th}, \\ & c_4 > 2c_3 - c_2 - 2\tau_d \\ 2E\left[\frac{-g_-(t_{1,2},t_{1,4})}{\tau_-(1+g_-(t_{1,2},t_{1,4})}\right] & \text{if } v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,5}+\tau_d,3,5) < V_{th} \\ 2E\left[\frac{-g_+(t_{1,1},t_{1,4})}{\tau_+(1+g_+(t_{1,1},t_{1,4})} + \frac{-g_+(t_{1,2},t_{1,4})}{\tau_+(1+g_+(t_{1,2},t_{1,4})} + \right. & \\ \left. \frac{-g_+(t_{1,3},t_{1,4})}{\tau_+(1+g_+(t_{1,3},t_{1,4})} + \frac{g_-(t_{1,4},t_{1,5})}{\tau_-(1+g_-(t_{1,4},t_{1,5})}\right] & \text{if } v(t_{1,4}+\tau_d,1,4) > V_{th} \\ 2E\left[\frac{g_+(t_{1,4},t_{1,5})}{\tau_+(1+g_+(t_{1,4},t_{1,5})}\right] & \text{if } v(t_{1,5}+\tau_d,1,5) > V_{th} \end{cases} \tag{A.4}$$

$$\frac{\partial E^2}{\partial c_5} = \begin{cases} 0 & \text{if } v(t_{1,3}+\tau_d,1,3) > V_{th}, \\ & v(t_{1,5}+\tau_d,4,5) > V_{th}, \\ & c_5 > 2c_4 - c_3 - 2\tau_d \\ 2E\left[\frac{-g_+(t_{1,4},t_{1,5})}{\tau_+(1+g_+(t_{1,4},t_{1,5})}\right] & \text{if } v(t_{1,3}+\tau_d,1,3) > V_{th}, \\ & v(t_{1,5}+\tau_d,4,5) > V_{th}, \\ & c_5 < 2c_4 - c_3 - 2\tau_d \\ 2E\left[\frac{-g_+(t_{1,3},t_{1,5})}{\tau_+(1+g_+(t_{1,3},t_{1,5})} + \frac{-g_+(t_{1,4},t_{1,5})}{\tau_+(1+g_+(t_{1,4},t_{1,5})}\right] & \text{if } v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,5}+\tau_d,3,5) > V_{th}, \\ & c_5 < 2c_3 - c_2 - 2\tau_d \\ 2E\left[\frac{-g_+(t_{1,4},t_{1,5})}{\tau_+(1+g_+(t_{1,4},t_{1,5})}\right] & \text{if } v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,5}+\tau_d,3,5) > V_{th}, \\ & c_5 > 2c_3 - c_2 - 2\tau_d \\ 2E\left[\frac{-g_-(t_{1,4},t_{1,5})}{\tau_-(1+g_-(t_{1,4},t_{1,5})}\right] & \text{if } v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,4}+\tau_d,3,4) > V_{th} \\ 2E\left[\frac{-g_-(t_{1,2},t_{1,5})}{\tau_-(1+g_-(t_{1,2},t_{1,5})}\right] & \text{if } v(t_{1,2}+\tau_d,1,2) > V_{th}, \\ & v(t_{1,5}+\tau_d,3,5) < V_{th} \\ 2E\left[\frac{-g_-(t_{1,3},t_{1,5})}{\tau_-(1+g_-(t_{1,3},t_{1,5})}\right] & \text{if } v(t_{1,3}+\tau_d,1,3) > V_{th}, \\ & v(t_{1,5}+\tau_d,4,5) < V_{th} \\ 2E\left[\frac{-g_-(t_{1,4},t_{1,5})}{\tau_-(1+g_-(t_{1,4},t_{1,5})}\right] & \text{if } v(t_{1,4}+\tau_d,1,4) > V_{th} \\ 2E\left[\frac{-g_+(t_{1,1},t_{1,5})}{\tau_+(1+g_+(t_{1,1},t_{1,5})} + \frac{-g_+(t_{1,2},t_{1,5})}{\tau_+(1+g_+(t_{1,2},t_{1,5})} + \right. & \\ \left. \frac{-g_+(t_{1,3},t_{1,5})}{\tau_+(1+g_+(t_{1,3},t_{1,5})} + \frac{-g_+(t_{1,4},t_{1,5})}{\tau_+(1+g_+(t_{1,4},t_{1,5})}\right] & \text{if } v(t_{1,5}+\tau_d,1,5) > V_{th} \end{cases} \tag{A.5}$$

Bibliography

Abbott, L. and Kepler, T. B. (1990), "Model neurons: From hodgkin-huxley to hopfield," in *Statistical mechanics of neural networks*, pp. 5–18, Springer.

Albus, J. S. (1971), "A theory of cerebellar function," *Mathematical Biosciences*, 10, 25–61.

Andreou, A. G., Meitzler, R. C., Strohbehn, K., and Boahen, K. (1995), "Analog VLSI neuromorphic image acquisition and pre-processing systems," *Neural Networks*, 8, 1323–1347.

Banghart, M., Borges, K., Isacoff, E., Trauner, D., and Kramer, R. H. (2004), "Light-activated ion channels for remote control of neuronal firing," *Nature neuroscience*, 7, 1381–1386.

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

Bliss, T. V. and Gardner-Medwin, A. (1973), "Long-lasting potentiation of synaptic transmission in the dentate area of the unanaesthetized rabbit following stimulation of the perforant path," *The Journal of physiology*, 232, 357.

Bliss, T. V. and Lømo, T. (1973), "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, 331–356.

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

Borghetti, J., Snider, G. S., Kuekes, P. J., Yang, J. J., Stewart, D. R., and Williams, R. S. (2010), "Memristiveswitches enable statefullogic operations via material implication," *Nature*, 464, 873–876.

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

Burgsteiner, H. (2006), "Imitation learning with spiking neural networks and real-world devices," *Engineering Applications of Artificial Intelligence*, 19, 741–752.

Burkitt, A. (2006), "A Review of the Integrate and Fire Neuron Model: I. Homogeneous Synaptic Input," *Biol Cybern*, 95, 1–9.

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

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

Dora, S., Suresh, S., and Sundararajan, N. (2014), "A sequential learning algorithm for a Minimal Spiking Neural Network (MSNN) classifier," in *Neural Networks (IJCNN), 2014 International Joint Conference on*, pp. 2415–2421, IEEE.

Esser, S. K., Andreopoulos, A., Appuswamy, R., Datta, P., Barch, D., Amir, A., Arthur, J., Cassidy, A., Flickner, M., Merolla, P., et al. (2013), "Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pp. 1–10, IEEE.

Feichtinger, H. G. (2010), "Approximate reconstruction of bandlimited functions for the integrate and fire sampler," *Advanced Computational Mathematics*, p. 12.

Ferrari, S., Mehta, B., Muro, G. D., VanDongen, A. M., and Henriquez, C. (2008a), "Biologically Realizable Reward-Modulated Hebbian Training for Spiking Neural Networks," *Proc. International Joint Conference on Neural Networks, Hong Kong*, pp. 1781–1787.

Ferrari, S., Mehta, B., Di Muro, G., VanDongen, A. M., and Henriquez, C. (2008b), "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*, pp. 1780–1786, IEEE.

Ferster, D. and Spruston, N. (1995), "Cracking the neuronal code," *SCIENCE-NEW YORK THEN WASHINGTON-*, pp. 756–756.

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

FitzHugh, R. (1961), "Impulses and physiological states in theoretical models of nerve membrane," *Biophysical journal*, 1, 445–466.

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

Florian, R. V. (2007a), "Reinforcement Learning Through Modulation of Spike-Timing-Dependent Synaptic Plasticity," *Neural Computation*, 19, 1468–1502.

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

Foderaro, G., Henriquez, C., and Ferrari, S. (2010), "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*, pp. 911–917, IEEE.

Gerstner, W. and Kistler, W. (2006), *Spiking Neuron Models: Single Neurons, Populations, Plasticity*, Cambridge University Press, Cambridge, UK.

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

Gerstner, W., Kempter, R., van Hemmen, J. L., and Wagner, H. (1999), "Hebbian learning of pulse timing in the Barn Owl auditory system, Pulsed neural networks," .

Harrison, R. R. and Koch, C. (1998), "An analog VLSI model of the fly elementary motion detector," *Advances in neural information processing systems*, pp. 880–886.

Hodgkin, A. L. and Huxley, A. F. (1952a), "A Quantitative Description of Ion Currents and its Applications to Conductance and Excitation in Nerve Membranes," *Journal of Physiology*, 117, 500–544.

Hodgkin, A. L. and Huxley, A. F. (1952b), "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, 117, 500–544.

Hubel, D. H. and Wiesel, T. N. (1959), "Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology*, 148, 574–591.

Indiveri, G. (1998), "Analog VLSI model of locust DCMD neuron response for computation of object approach," *PROGRESS IN NEURAL PROCESSING*, 10, 47–60.

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

Jack, J. J. B., Nobel, D., and Tsien, R. (1975), *Electric Current Flow in Excitable Cells, 1st ed.*, Oxford University Press, Oxford, UK.

Jo, S. H., Chang, T., Ebong, I., Bhadviya, B., Mazumder, P., and Lu, W. (2010), "Nanoscale Memristor Device as Synapse in Neuromorphic Systems," *Nano Letters*, 10, 1297–1301.

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

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

Kim, H., Sah, M. P., Yang, C., Roska, T., and Chua, L. O. (2012), "Neural synaptic weighting with a pulse-based memristor circuit," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 59, 148–158.

Kuhn, K. J. (2009), "Moore's Law Past 32nm: Future Challenges in Device Scaling," in *Computational Electronics, 2009. IWCE'09. 13th International Workshop on*, pp. 1–6, IEEE.

LaValle, S. M. (2004), "Planning Algorithms," .

Legenstein, R., Naeger, C., and Maass, W. (2005), "What Can a Neuron Learn with Spike-Timing-Dependent Plasticity?" *Neural Computation*, 17, 2337–2382.

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

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

Lichtsteiner, P., Posch, C., and Delbruck, T. (2008), "A 128× 128 120 dB 15 μs latency asynchronous temporal contrast vision sensor," *Solid-State Circuits, IEEE Journal of*, 43, 566–576.

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

Maass, W. (1997b), "Noisy Spiking Neurons with Temporal Coding Have More Computational Power than Sigmoidal Neurons," *Advances in Neural Information Processing Systems*, 9, 211–217.

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

Markram, H., Lübke, J., Frotscher, M., and Sakmann, B. (1997), "Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs," *Science*, 275, 213–215.

Mead, C. A. and Mahowald, M. A. (1988), "A silicon model of early visual processing," *Neural networks*, 1, 91–97.

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

Morris, C. and Lecar, H. (1981), "Voltage oscillations in the barnacle giant muscle fiber," *Biophysical journal*, 35, 193–213.

Mountcastle, V. B. (1957), "Modality and topographic properties of single neurons of cat's somatic sensory courtex," *J. neurophysiol*.

Nagumo, J., Arimoto, S., and Yoshizawa, S. (1962), "An active pulse transmission line simulating nerve axon," *Proceedings of the IRE*, 50, 2061–2070.

Pennartz, C. (1997a), "Reinforcement learning by Hebbian synapses with adaptive thresholds," *Neuroscience*, 81, 303–319.

Pennartz, C. M. A. (1997b), "Reinforcement Learning by Hebbian Synapses with Adaptive Thresholds," *Neuroscience*, 81, 303–319.

Pfeiffer, K. and Homberg, U. (2014), "Organization and functional roles of the central complex in the insect brain," *Annual review of entomology*, 59, 165–184.

Pfister, J.-P., Barber, D., and Gerstner, W. (2003), "Optimal Hebbian learning: a probabilistic point of view," in *Artificial Neural Networks and Neural Information ProcessingICANN/ICONIP 2003*, pp. 92–98, Springer.

Pfister, J. P., Toyoizumi, T., Barber, D., and Gerstner, W. (2006), "Optimal Spike-Timing-Dependent Plasticity for Precise Action Potential Firing in Supervised Learning," *Neural Computation*, 18, 1318–1348.

Ponulak, F. and Kasinski, A. (2010), "Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting," *Neural Computation*, 22, 467–510.

Rostro-Gonzalez, H., Cessac, B., and Viéville, T. (2012), "Parameter estimation in spiking neural networks: a reverse-engineering approach," *Journal of neural engineering*, 9, 026024.

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

Saleh, A. Y., Hameed, H. N. B. A., Najib, M., and Salleh, M. (2014), "A Novel hybrid algorithm of Differential evolution with Evolving Spiking Neural Network for pre-synaptic neurons Optimization," *Int. J. Advance Soft Compu. Appl*, 6.

Schutter, E. D. (2009), *Computational Modeling Methods for Neuroscientists*, The MIT Press, 1st edn.

Shin, S., Kim, K., and Kang, S. (2011), "Memristor applications for programmable analog ICs," *Nanotechnology, IEEE Transactions on*, 10, 266–274.

Sjostrom, P., Turrigiano, G., and Nelson, S. (2001), "Rate, Timing, and Cooperativity Jointly Determine Cortical Synaptic Plasticity," *Neuron*, 32, 1149 – 1164.

Song, S., Miller, K., and Abbott, L. (2000), "Competitive Hebbian Learning through spike-timing-dependent synaptic plasticity," *Nature*, 3, 919–926.

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

Stengel, R. F. (1986), *Optimal Control and Estimation*, Dover Publications, Inc.

Strausfeld, N. J. (2012), *Arthropod brains: evolution, functional elegance, and historical significance*, Belknap Press of Harvard University Press Cambridge, MA.

Strausfeld, N. J. and Hirth, F. (2013), "Deep homology of arthropod central complex and vertebrate basal ganglia," *Science*, 340, 157–161.

Strauss, R. (2002), "The central complex and the genetic dissection of locomotor behaviour," *Current opinion in neurobiology*, 12, 633–638.

Strukov, D. B., Snider, G. S., Stewart, D. R., and Williams, R. S. (2008), "The missing memristor found," *nature*, 453, 80–83.

Thorpe, S., Delorme, A., and Van Rullen, R. (2001), "Spike-based strategies for rapid processing," *Neural networks*, 14, 715–725.

Van De Ven, T. J., VanDongen, H. M. A., and VanDongen, A. M. J. (2005), "The non-kinase phorbol ester receptor alpha 1-chimerin binds the NMDA receptor NR2A subunit and regulates dendritic spine density." *Journal of Neuroscience*, 15, 9488–9496.

Wang, J., Belatreche, A., Maguire, L., and McGinnity, T. (2014), "An Online Supervised Learning Method for Spiking Neural Networks with Adaptive Structure," *Neurocomputing*.

Wong, H.-S. P., Frank, D. J., Solomon, P. M., Wann, C. H., and Welser, J. J. (1999), "Nanoscale cmos," *Proceedings of the IEEE*, 87, 537–570.

Wysoski, S. G., Benuskova, L., and Kasabov, N. (2006), "Adaptive learning procedure for a network of spiking neurons and visual pattern recognition," *Advanced Concepts for Intelligent Vision Systems, ACIVS, Antwerp, Lecture Notes in Computer Science*, 4179.

Yakopcic, C., Taha, T. M., Subramanyam, G., Shin, E., Murray, P. T., and Rogers, S. (2010), "Memristor-based pattern recognition for image processing: an adaptive coded aperture imaging and sensing opportunity," in *SPIE Optical Engineering+ Applications*, pp. 78180E–78180E, International Society for Optics and Photonics.

Yang, Z., Murray, A., Worgotter, F., Cameron, K., and Boonsobhak, V. (2006), "A neuromorphic depth-from-motion vision model with STDP adaptation," *Neural Networks, IEEE Transactions on*, 17, 482–495.

Zemelman, B. V., Lee, G. A., Ng, M., and Miesenböck, G. (2002), "Selective photostimulation of genetically chARGed neurons," *Neuron*, 33, 15–22.

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

Zhang, X. and Xu, Z. (2013), "navigation of virtual insect in different terrains," .

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

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