12-2020

Towards Sensorimotor Coupling of a Spiking Neural Network and Deep Reinforcement Learning for Robotics Application

Kashu Yamazaki

TOWARDS SENSORIMOTOR COUPLING OF A SPIKING NEURAL NETWORK AND DEEP REINFORCEMENT LEARNING FOR ROBOTICS APPLICATION

KASHU YAMAZAKI

An Honors Thesis submitted in partial fulfillment of the requirements for the Honors Program in Mechanical Engineering

Mechanical Engineering
University of Arkansas

Dec 2020 – Submission

This thesis is dedicated to everyone who has supported me through my undergraduate degree at the University of Arkansas. Special mention goes out to my supervisor, Dr. Ngan Le, and to my lab-mates. I greatly appreciate and acknowledge the support, friendships and collaboration received during my undergraduate career through our work and numerous inspiring conversations. I learned much from each of them all and I'll always be grateful for the privilege of working with them. My thanks also goes to my family and friends, whose patient support made this work possible. I salute you all for the selfless love, care, pain and sacrifice you did to shape my life.

2016 – 2020

ABSTRACT

Deep reinforcement learning augments the reinforcement learning framework and utilizes the powerful representation of deep neural networks. Recent works have demonstrated the great achievements of deep reinforcement learning in various domains including finance, medicine, healthcare, video games, robotics and computer vision. Deep neural network was started with Multi-layer Perceptron (1^{st} generation) and developed to deep neural networks (2^{nd} generation) and it is moving forward to spiking neural networks which are known as 3^{rd} generation of neural networks. Spiking neural networks aim to bridges the gap between neuroscience and machine learning, using biologically-realistic models of neurons to carry out computation. In this thesis, we first provide a comprehensive review on both spiking neural networks and deep reinforcement learning with emphasis on robotic applications. Then we will demonstrate how to develop a robotics application for context-aware scene understanding to perform sensorimotor coupling. Our system contains two modules corresponding to scene understanding and robotic navigation. The first module is implemented as a spiking neural network to carry out semantic segmentation to understand the scene in front of the robot. The second module provides a high-level navigation command to robot, which is considered as an agent and implemented by online reinforcement learning. The module was implemented with biologically plausible local learning rule that allows the agent to adopt quickly to the environment. To benchmark our system, we have tested the first module on Oxford-IIIT Pet dataset and the second module on the custom made Gym environment. Our experimental results have proven that our system is able present the competitive results with deep neural network in segmentation task and adopts quickly to the environment.

keywords: *Spiking Neural Networks, Online Learning, Robotics*

PUBLICATIONS

Some ideas and figures may have appeared previously in the following publications:

Published papers

Le, N., Yamazaki, K., Truong, D., Gia Quach, K., and Savvides, M., "A Multi-task Contextual Atrous Residual Network for Brain Tumor Detection Segmentation", accepted by International Conference on Pattern Recognition (ICPR), 2021.

Yamazaki, K., Le, N., Rathour, V., "Invertible Residual Network with Regularization for Effective Volumetric Segmentation", accepted by SPIE Medical Imaging, 2021.

Le, N., Le, T., Yamazaki, K., Bui, T. D., Luu, K., and Savides, M., "Offset Curves Loss for Imbalanced Problem in Medical Segmentation", accepted by International Conference on Pattern Recognition (ICPR), 2021.

Liu, Y., Yamazaki, K., Zhang, D., Li, Y., Su, M., Xie, Q., Chen, Y., and Bai. M., "Minimally Invasive Intraperitoneal Photodynamic Therapy using a New Soft Robot System ", SPIE 11220, Optical Methods for Tumor Treatment and Detection: Mechanisms and Techniques in Photodynamic Therapy XXIX, 2020.

Sirotkin, E., Yamazaki, K., and Miroshnichenko., A., "Gearbox Development for an Emergency Brake System of the Wind Turbine ", IOP Conference Series: Earth and Environmental Science, Volume 459, Chapter 1., 2020.

Papers under review

Le, N., Rathor, V., Yamazaki, K., Luu, K., Savvides, M., "Deep Reinforcement Learning in Computer Vision: A Comprehensive Survey ", Artificial Intelligence Review.

ACKNOWLEDGMENTS

I would like to express my great appreciation to Dr. Ngan Le for her patient guidance and great support during my senior year, especially for preparing myself in computer vision and artificial intelligence. Her guidance, expertise, and understanding have helped me grow as a researcher, while also giving me valuable experience. I would also like to acknowledge all of the support from Dr. Zhou and Dr. Roe.

Regarding LyX: Further thanks to *Nicholas Mariette* and *Ivo Pletikosić*for producing a LyXport.

CONTENTS

LIST OF TABLES

1 INTRODUCTION

1.1 MOTIVATION

The biological intelligence of living creatures allows them to autonomously perform complex tasks in a dynamic environment. Such intelligence basically depends on embodiment, and the interaction of self with surrounding environment is indispensable to intellectual development [6]. Through the long history of evolution, biology acquired remarkable capability in sensory-motor coupling – the dynamic integration of the sensory system and motor system for creatures to take sensory information and make full use of it for motor actions, which results in their elegant behaviour. It should be specially mentioned that such sensorimotor integration is modified over time by internal and external variables to adopt themselves in the ever changing environment. With the increasing needs for autonomy of machines in the real world, e.g. self-driving vehicles, drones, and collaborative industrial robots, advent of control strategies acquired by living creatures, which are energy and computationally efficient with real-time self-learning capability is anticipated. This is especially important for applications in embedded systems and robotics because real-time responses are crucial and energy supply is limited in those applications.

Currently, neither classical control strategies nor conventional artificial neural networks, including deep neural networks (DNNs) can meet those needs [19]. Even though DNNs have succeeded in capturing some essential information processing stages in perception such as feature extraction from high-dimensional sensory data [87] and demonstrated state-of-the-art results in many applications, they still suffer from some fundamental drawbacks mainly due to the reliance on the successful back-propagation algorithm. They are sample and memory inefficient[1] and unsuitable for online or incremental learning from data streams. A promising solution to this previously infeasible applications could be given by biologically plausible spiking neural networks (SNNs)[2]. Due to their functional similarity to the biological neural networks, SNNs can embrace the sparsity found in biology and highly compatible with temporal code. Although SNNs still lag

1 Some studies tries to reduce the memory consumption during training by introducing reversible architectures, where each layer's activations can be reconstructed exactly from the next layer's. Therefore, the activations for most layers need not be stored in memory during backpropagation [23, 45, 55].

2 The energy efficiency is expected in SNNs partly due to the algorithms, but mostly due to the hardware, called Neuromorphic Hardware including but not limited to TrueNorth [106], Loihi [36], SpiNNaker [49], NeuroGrid [14], and DYNAPs [113].

behind DNNs in terms of accuracy, the gap is decreasing, and can even vanish on some tasks, while SNNs typically require fewer operations. However, SNNs are difficult to train in general, mainly owing to their complex dynamics of neurons and non-differentiable spike operations. In this thesis, we will explore the ability of SNNs with emphasis on robotics control. To validate our SNN based models, we have also developed a simulation environment, which can be used for safe exploration and validation of strategy before implemented on the actual hardware.

1.2 THE STRUCTURE OF THIS THESIS

The remainder of this thesis is structured as follows. Chapter 2 provides background and context of the research by reviewing theoretical background of SNNs as well as DNNs in a context of robotic control. Chapter 3 presents the techniques, models and methodology that was used in this project. Chapter 4 presents the experiments performed to validate our model.

2 BACKGROUND

2.1 AN OVERVIEW OF BIOLOGICAL NEURONS

Approximately 86 billion neurons can be found in the human nervous system [7]. Neurons are the basic working units of the nervous system and communicate with other neurons via projections from the cell body. Basically, the input part of the neurite is called the dendrite and the output part is called the axon.A typical structure of neuron is shown in Figure 2.4. Each neuron receives an input signal from the dendrites and generates an output signal (action potential) along the axon when the membrane potential of the neuron reaches a threshold. At the axon terminals, the arrival of an action potential causes synaptic vesicles to be released from their protein anchors, allowing the vesicles to fuse with the cell membrane and release neurotransmitters into the synaptic cleft. The site where information is exchanged between neurites is called a synapse.

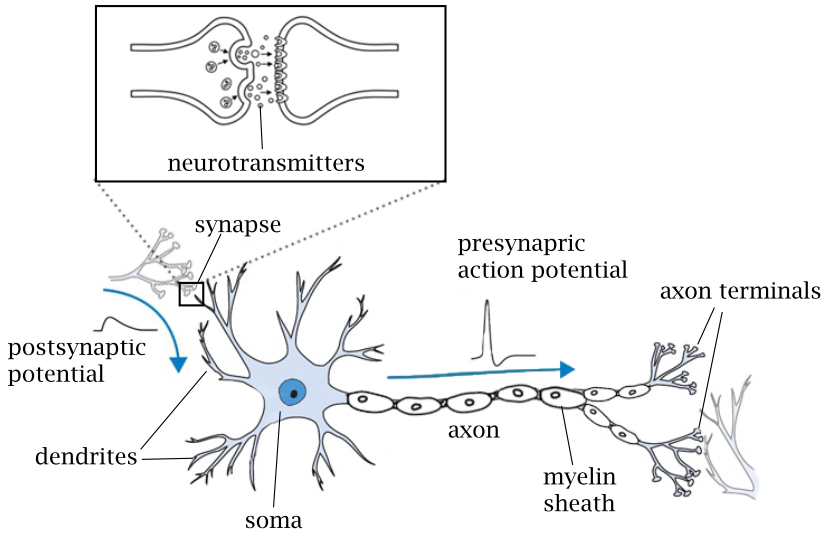


Figure 2.1: Structure of a typical biological neuron and synapse.

2.1.1 *Reversal Potential*

The specific ionic composition of the cytosol usually differs greatly from that of the extracellular fluid. Because of the concentration difference between inside and outside of a cell, those ions tends to move in the direction which leads to the concentration equilibrium, creating

ION	MAMMALIAN BLOOD $[mM]$	MAMMALIAN CELL $[mM]$
K^+	5.5	150
Na^+	135	15
Cl^-	125	9
Ca^{2+}	1.8	10^{-4}

Table 2.1: Typical ion concentrations in vertebrates

electromotive force. However, the movement of ions will be hindered by an electrical potential caused by the ions themselves. This leads to the equilibrium state, and the magnitude of equilibrium potential for ion A is given by the Nernst equation as follows:

$$E_A = \frac{RT}{zF} \ln \frac{[A]_{out}}{[A]_{in}} \tag{2.1}$$

where R is the universal gas constant ($= 8.31\ [JK^{-1}mol^{-1}]$), T is the absolute temperature 310.15 $[K]$ at human body temperature (37 [°C]), F is the Faraday constant ($= 96485\ [C \cdot mol^{-1}]$), z is the number of elementary charges of the ion in question involved in the reaction, $[A]_{out}$ is the extracellular concentration of ion A, and $[A]_{in}$ is the intracellular concentration of ion A.

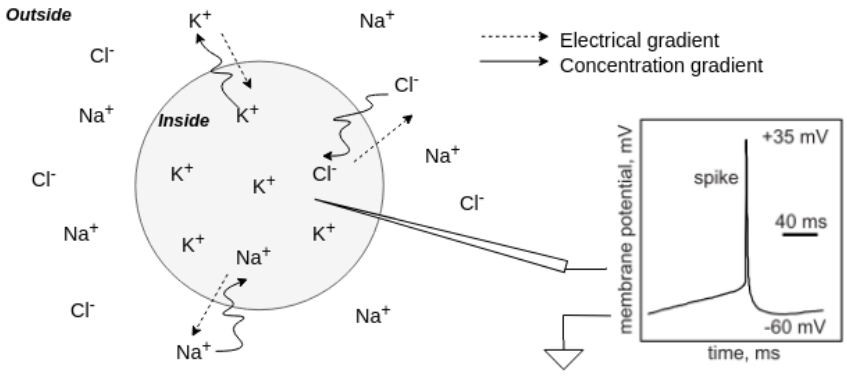


Figure 2.2: The illustration of ion concentration in a typical neuron.

The ion concentration in and outside of typical mammalian neurons are listed in Table 2.1. Based on the values on the table, reversal potentials can be estimated as follows:

$$E_K = 26.7 \ln \frac{5.5}{150} = -88.27\ [mV] \tag{2.2}$$

$$E_{Na} = 26.7 \ln \frac{135}{15} = 58.67\ [mV] \tag{2.3}$$

$$E_{Cl} = -26.7 \ln \frac{125}{9} = -70.25\,[mV] \tag{2.4}$$

2.1.2 *Membrane Potential*

The electric potential inside a cell with respect to the outside of the cell is called membrane potential v_m $[mV]$. The aforementioned Nernst equation derives the point at which the flow of each ion apparently stops, but the membrane potential itself cannot be calculated. The membrane potential can be derived using Goldman-Hodgkin-Katz equation, which takes into consideration the relative permeability of the plasma membrane to each ion in question[1].

$$v_m = \frac{RT}{F} \ln \frac{P_K[K^+]_{out} + P_{Na}[Na^+]_{out} + P_{Cl}[Cl^-]_{in}}{P_K[K^+]_{in} + P_{Na}[Na^+]_{in} + P_{Cl}[Cl^-]_{out}} \tag{2.5}$$

where P_A is the membrane permeability for ion A, and for a typical neuron at rest, it is known that $P_K : P_{Na} : P_{Cl} = 1 : 0.04 : 0.45$. In contrast, approximate relative permeability at the peak of a typical neuronal action potential are $P_K : P_{Na} : P_{Cl} = 1 : 12 : 0.45$ [].

Resting Membrane Potential

Due to the action of a number of proteins, ions are constantly moving in and out of the cell. Although influx of ions does not stop, charge transfer becomes apparently immobile when the total charge of the outflowing ions and the total charge of the inflowing ions per unit time becomes the same. The resting membrane potential of a cell is determined by the net flow of ions through the "leak" channels that are open in the resting state. Based on the relative membrane permeability for a typical neuron at rest, we can calculate the resting membrane potential E_m as follows[2]:

$$\begin{aligned} E_m &= \frac{RT}{F} \ln \frac{5.5P_K + 135 \times 0.04P_K + 9 \times 0.45P_K}{150P_K + 15 \times 0.04P_K + 125 \times 0.45P_K} \\ &= -70.15\,[mV] \end{aligned} \tag{2.6}$$

Since the reversal potential for Cl^- ion is typically close to the resting membrane potential, Cl^- ion is usually ignored when discussing a neuron's resting membrane potential.

1 Given the condition that the permeability of all but one of the ions is zero, Goldman equation is consistent with the Nernst equation.

2 Note that $P_{Na} = 0.04P_K$ and $P_{Cl} = 0.45P_K$

2.1.3 *Action Potential*

When an action potential occurs, sodium channels on the axon are opened and Na^+ ions are free to move in and out of the cell membrane. Therefore, the membrane potential fluctuates towards the reversal potential of Na^+ ion. The sodium channel is then inactivated and closed, and now the potassium channel, which is potential-dependent, is opened. Now, the membrane potential descends toward the reversal potential of K^+ ion and undershoots beyond the resting membrane potential E_m.

$$\begin{aligned} v_{peak} &= \frac{RT}{F} \ln \frac{5.5P_K + 135 \times 12P_K + 9 \times 0.45P_K}{150P_K + 15 \times 12P_K + 125 \times 0.45P_K 0} \\ &= 38.43\,[mV] \end{aligned} \tag{2.7}$$

2.1.4 *Synapse*

A synapse is a contact structure for information transfer that develops between the side that outputs neural information and the side that inputs it. The most basic structure is one in which the axon ends of presynaptic cells contact the dendrites of postsynaptic cells. Synapses can be broadly divided into chemical and electrical synapses. At chemical synapses, which occupy many synapses in the central nervous system, the arrival of action potentials opens voltage-gated calcium channels in the pre-synaptic cell, resulting in calcium influx and the opening release of synaptic granules. As a result, neurotransmitters contained in the synaptic granules are released into the synaptic cleft. Neurotransmitters bind to neurotransmitter receptors in the post-synaptic cell and transmit by directly altering membrane potential or by activating intracellular secondary messengers. Chemical synapses are subdivided into excitatory and inhibitory synapses. Electrical synapses, on the other hand, are structures that transmit membrane potential changes directly to the next neuron via gap junctions on the contact membrane. The synaptic potential received in this way travels to the cell body and is integrated at the axonal nodule, which ultimately determines whether or not the post-synaptic cell fires. This interaction of influences is called neural integration. Moreover, the efficiency of synaptic transmission is not necessarily constant, but varies with the intensity of the input. This is called synaptic plasticity and is thought to be a cellular mechanism of learning and memory.

Excitatory Synapse

Excitatory synapses are synaptic connections that depolarize postsynaptic cells through synaptic transmission and promote the firing of action potentials.

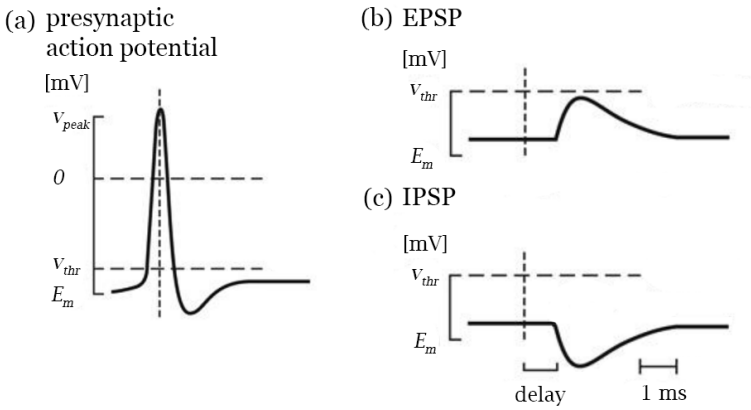


Figure 2.3: (a) presynaptic action potential. (b) excitatory postsynaptic potential (EPSP) (c) inhibitory postsynaptic potential (IPSP)

Binding of neurotransmitters to ion channel-coupled receptors at excitatory synapses increases the conductance of cations. Because the membrane potential of the cell is lower than the reversal potential of these receptors near the resting membrane potential, extracellular cations such as Na^+ and Ca^{2+} flow into the postsynaptic cell and depolarize the membrane potential. This change in membrane potential is called excitatory postsynaptic potential (EPSP). At this time, the current flows toward the inside of the cell, and this inward-facing current is called excitatory postsynaptic current (EPSC).

Inhibitory Synapse

Inhibitory synapses are synaptic connections that hyperpolarize postsynaptic cells by synaptic transmission and inhibit the development of action potentials.

2.2 RATE-BASED NEURON MODEL

Since the firing rate of neurons in the primary motor cortex (M1) or primary visual cortex (V1) correlates with motion or presentation stimulus, the firing rate of those neurons was thought to be a feature that describes such information. In this point of view, activity of a neuron is represented only by the macroscopic feature, firing rate r, regardless of change in membrane potential or spike timing.

The first rate-coded artificial neuron, which is known as formal neuron (or threshold logic unit), was proposed by Mcculloch and Pitts in 1943 [105]. Based on the formal neuron, Rosenblatt introduced perceptron[3] in 1958 [138]. These first generation neurons fire binary signals when the sum of incoming signals reaches a threshold of a

3 In the context of neural networks, a perceptron refers to an artificial neuron using the Heaviside step function as the activation function.

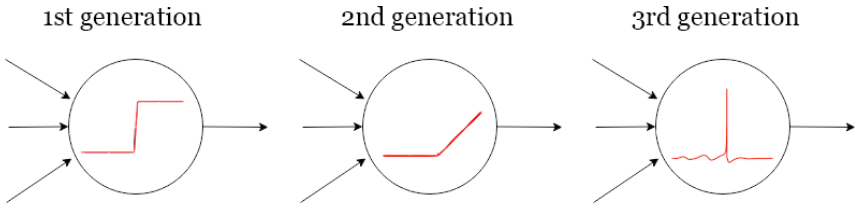


Figure 2.4: Three generations of artificial neuron models. Each models are characterized by: binary activation function, continuous (mostly) differentiable activation function, temporally sparse activation function.

neuron. Since they are capable of performing mathematical operations with boolean outputs, they have been applied to neural networks such as multi-layer perceptrons.

This concept is later extended to utilize continuous activation functions including the sigmoid [58] or hyperbolic tangent function to deal with analog (continuous) inputs and outputs; consequently, this enabled to train the neural network through a powerful backpropagation algorithm that exploits gradient-descent. Because of the proven ability of a sufficiently large neural network of artificial neurons to approximate any analog function arbitrarily well[4], artificial neural networks have been widely used as a powerful information-processing tool in machine learning. The analog information signals used in the second generation neurons can be interpreted as an abstracted rate coding. In general, the discrete-time firing rate models can be formulated as follows:

$$\mathbf{r} = f(\mathbf{W}\mathbf{u} + \mathbf{b}) \tag{2.8}$$

where $\mathbf{u} \in \mathbb{R}^{N_{pre}}$ is the firing rate of presynaptic neurons, $\mathbf{r} \in \mathbb{R}^{N_{post}}$ is the firing rate of postsynaptic neurons, $\mathbf{W} \in \mathbb{R}^{N_{post} \times N_{pre}}$ is the weight matrix that represents the synaptic strength between the pre- and postsynaptic neurons, $\mathbf{b} \in \mathbb{R}^{N_{post}}$ is the bias term, and $f(\cdot)$ is the non-linear activation function[5].

When the output firing rate does not follow the input fluctuation immediately, but follows it with a time constant τ_r, the model can be expressed in a form of ordinary differential equation as follows:

$$\tau_r \frac{d\mathbf{r}}{dt} = -\mathbf{r} + f(\mathbf{W}\mathbf{u} + \mathbf{b}) \tag{2.9}$$

4 Universal approximation theorem states that a feed-forward network with a single hidden layer with a finite number of neurons can approximate continuous functions, under assumptions on the non-polynomial activation function [34, 92]. Sigmoidal activation function and the ReLU are also proved to follow the universal approximation theorem [152].

5 Rectified Linear Unit (ReLU) [119] and its variants are commonly employed because they tend to show better convergence performance than sigmoidal activation function [83].

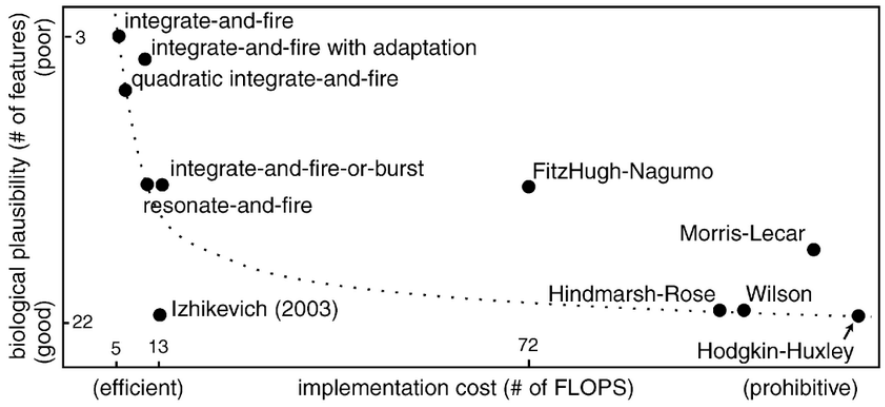


Figure 2.5: A comparison of spiking neuron models in terms of implementation cost and biological plausibility (adopted from [71]).

The second generation neurons do not model electrical pulses, which will be considered in spike-based neuron models.

2.3 SPIKE-BASED NEURON MODELS

The ability to simultaneously record the activity of multiple cells has led to the idea that the time difference between spikes in different neurons and the spike timing itself can have functional significance. Since the firing rate model cannot handle the problem of this perspective, a model describing the timing of spikes and the variation of the sub-threshold membrane potential is needed. The model that handles the generation of such spikes is distinguished from the firing rate model and called the spiking neuron model.

Spiking neurons are more closely modeled after the biological neurons and they can utilize temporal codes in their computations. Variety of spiking neuron models have been proposed, and they display trade-offs between biological accuracy and computational feasibility (Figure 2.5). Choosing an appropriate model depends on the user requirements. Spike-based neuron models are reviewed regarding the computational efficiency and biological plausibility in [71]. In this section several models of spiking neurons are presented. Note that models of synapse will be presented in Section 2.4.

2.3.1 *Leaky Integrate and Fire (LIF) Model*

The model in which the input current is integrated over time until the membrane potential reaches a threshold without taking into account the biological ion channel behavior is called the Integrate-and-fire (IF) model[6] [2]. The typical response of IF model to the step input is shown in Figure 2.6a. Leaky integrate-and-fire (LIF) model reflects

6 IF model is just the time derivative of the law of capacitance, $Q = C_m v_m$

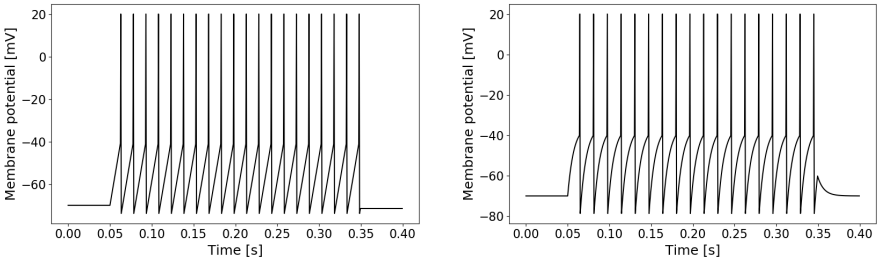
the diffusion of ions that occurs through the membrane when some equilibrium is not reached in the cell by introducing a "leak" term to the IF model. Because of its simplicity and low computational cost, LIF model and its variants are one of the widely used instance of spiking neuron model. The model is represented by the following equations, where the first term on the RHS of Equation 2.10 is the "leak" term:

$$C_m \frac{dv_m(t)}{dt} = -\frac{1}{R_m}(v_m - E_m) + I_{syn}(t) \tag{2.10}$$

$$\text{if} \quad v_m \geq v_\theta \quad \text{then} \quad v_m \leftarrow v_{peak} \quad \text{then} \quad v_m \leftarrow v_{reset} \tag{2.11}$$

where C_m is membrane capacitance $[pF]$, v_m is the membrane potential $[mV]$, R_m is the membrane resistance $[M\Omega]$, E_m is the resting membrane potential $[mV]$, and I_{syn} is synaptic current input $[pA]$.

The model can be made more accurately by introducing a refractory period τ_{ref} that limits the firing frequency of a neuron by preventing it from firing during that period. The typical response of LIF model to the step input is shown in Figure 2.6b.



(a) Integrate and Fire (IF) neuron (b) Leaky Integrate and Fire (LIF) neuron

Figure 2.6: Response of IF and LIF models to a step synaptic drive. The step synaptic drive of 70 $[pA]$ was applied during 50 $\sim$ 350 $[ms]$.

2.3.2 *Hodgkin-Huxley Model*

Hodgkin and Huxley conducted the experiment on the giant axon of a squid and concluded that two types of ion channels, K^+ channel and Na^+ channel, are involved in the generation of the action potential [67]. This model can be expressed by adding two terms that take care of the behavior of those two ion channels to Equation 2.10. Although the change in permeability of the ion channel is actually due to the structural change of the protein, it can be described phenomenologically by the analogy of opening and closing the gates.

$$C_m \frac{dv_m(t)}{dt} = -I_{ion}(t) + I_{syn}(t) \tag{2.12}$$

$$I_{ion}(t) = \frac{n^4}{R_K}(v_m - E_K) + \frac{m^3 h}{R_{Na}}(v_m - E_{Na}) + \frac{1}{R_L}(v_m - E_L) \quad (2.13)$$

where E_K represents reversal potential of K^+ ion (Equation 2.2), E_{Na} represents reversal potential of Na^+ ion (Equation 2.3), and E_L represents leak reversal potential, which is now thought to be a Cl^- ion's reversal potential (Equation 2.4). n, m, and h are dimensionless quantities between 0 and 1 that are associated with potassium channel activation, sodium channel activation, and sodium channel inactivation, respectively.
The three gates, n, m, and h, are described by the following differential equation, where g represents the gating variables n, m, and h, and the transition rate[7] for each gate $\alpha_g(v)$ and $\beta_g(v)$ are defined in Equation 2.15[8].

$$\frac{dg}{dt} = \alpha_g(v_m)(1 - g) - \beta_g(v_m)g \quad (2.14)$$

$$\begin{aligned} \alpha_m(v_m) &= \tfrac{0.1(25-v_m)}{\exp((25-v_m)/10)-1} & \beta_m(v_m) &= 4\exp(-v_m/18) \\ \alpha_h(v_m) &= 0.07\exp(-v_m/20) & \beta_h(v_m) &= \tfrac{1}{\exp((30-v_m)/10)+1} \\ \alpha_n(v_m) &= \tfrac{0.01(10-v_m)}{\exp((10-v_m)/10)-1} & \beta_n(v_m) &= 0.125\exp(-v_m/80) \end{aligned} \quad (2.15)$$

By solving these equations, Hodgkin-Huxley model can simulate the membrane potential behavior during spike generation without introducing spike generation procedures presented in LIF model (Equation 2.11). Although Hodgkin-Huxley model is biologically accurate[9], it demands large computational resource and infeasible in large-scale simulations.

2.3.3 *Izhikevich Model*

Izhikevich proposed a model that combines the biologically plausibility of Hodgkin–Huxley model's dynamics and the computational efficiency of LIF neurons [72]. The model is represented by the two-

7 $\alpha_g(v)$ is the transition rate from non-permissive to permissive states whereas $\beta_g(v)$ is the transition rate from permissive to non-permissive states.

8 In neural simulation software packages, the rate constants in Hodgkin-Huxley models are often parameterized using a generic functional form [121]: $\frac{A+Bv_m}{C+H\exp(\frac{v_m+D}{F})}$

9 Hodgkin-Huxley model is limited in the way that it only describes the channels and flow of ions in the neuron when generating spikes. Several drawbacks have been pointed out [107, 155].

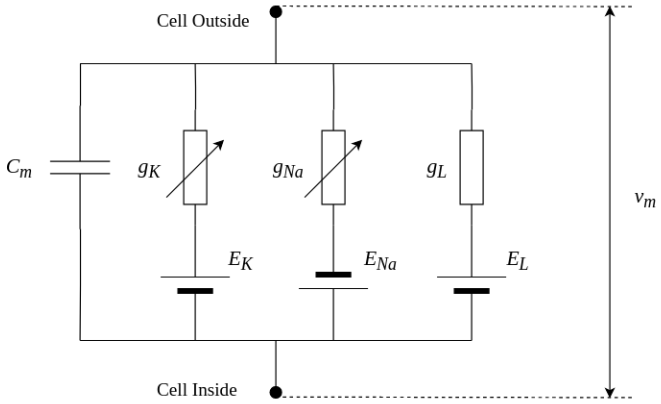


Figure 2.7: Illustration of electrical equivalent circuit, which was used by Hodgkin and Huxley [67] to mimic the electrical behavior of the membrane of the giant squid axon. The resistor with an arrow indicates voltage-dependent conductance.

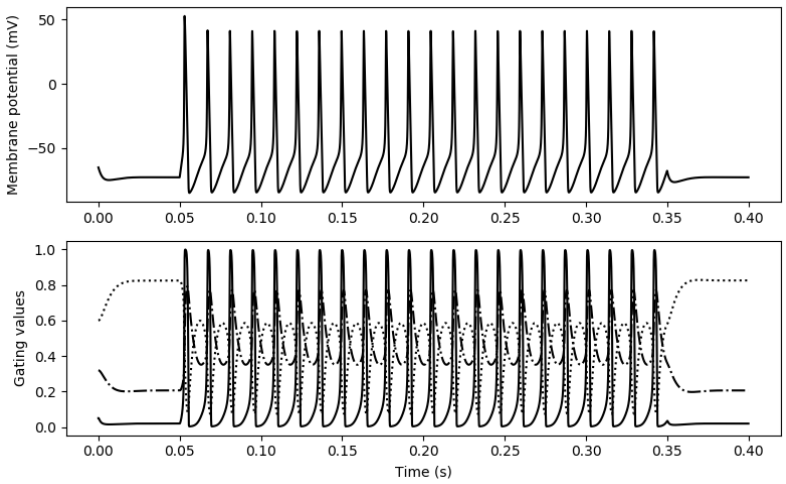


Figure 2.8: Response of Hodgkin-Huxley model (top) and behavior of gating values (bottom).

dimensional (2D) system of ordinary differential equations[10], and the Izhikevich model [73] can be expressed in the following form :

$$C_m \frac{dv_m(t)}{dt} = k(v_m - E_m)(v_m - v_t) - u + I_{syn}(t) \tag{2.16}$$

$$\frac{du(t)}{dt} = a(b(v_m - E_m) - u) \tag{2.17}$$

with the auxiliary after-spike resetting

$$\text{if } v_m \geq v_{peak} \text{ then } \begin{cases} v_m \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{2.18}$$

where v represents the membrane potential of a neuron $[mV]$ and u represents the activation of K^+ ionic currents and inactivation of Na^+ ionic currents $[pA]$. C_m is the membrane capacitance $[pF]$, E_m is the resting membrane potential $[mV]$, and v_t is the instantaneous threshold potential $[mV]$.

Izhikevich model can exhibit the firing patterns of all known types of cortical neurons with the choice of parameters based on [73]. Each type of neuron has following properties and membrane potential reacts to a step synaptic drive as shown in Figure 2.9.

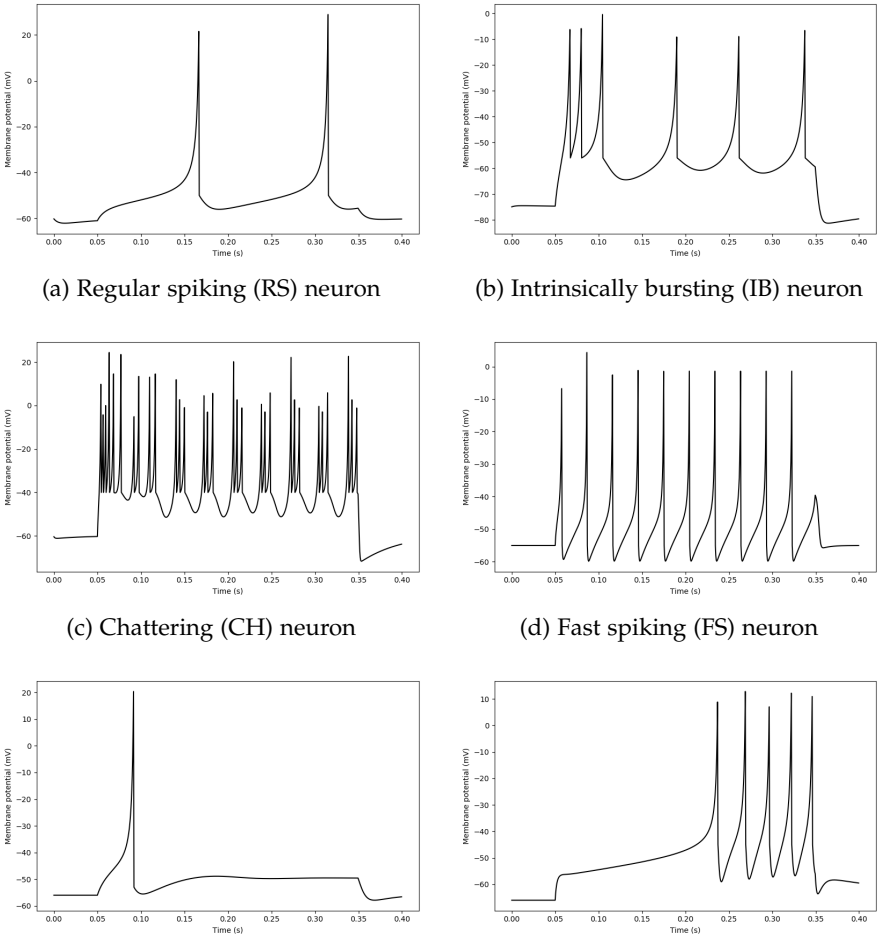
A. *Regular Spiking (RS) neuron:* (Figure 2.9a) Regular spiking neurons fire tonic spikes with adapting (decreasing) frequency in response to injected pulses of DC current. $100\dot{v_m} = 0.7(v_m + 60)(v_m + 40) - u + I$, $\dot{u} = 0.03(-2(v_m + 60) - u)$, if $v_m \geq 35$, then $v_m \leftarrow -50$, $u \leftarrow u + 100$.

B. *Intrinsically Bursting (IB) neuron:* (Figure 2.9b) Intrinsically bursting neurons generate a burst of spikes at the beginning of a strong depolarizing pulse of current, then switch to tonic spiking mode. $150\dot{v_m} = 1.2(v_m + 75)(v_m + 45) - u + I$, $\dot{u} = 0.01(5(v_m + 75) - u)$, if $v_m \geq 50$, then $v_m \leftarrow -56$, $u \leftarrow u + 130$.

C. *Chattering (CH) neuron:* (Figure 2.9c) Chattering neurons fire high-frequency bursts of spikes with relatively short inter burst periods. $50\dot{v_m} = 1.5(v_m + 60)(v_m + 40) - u + I$, $\dot{u} = 0.03((v_m + 60) - u)$, if $v_m \geq 25$, then $v_m \leftarrow -40$, $u \leftarrow u + 150$.

D. *Fast spiking (FS) neuron:* (Figure 2.9d) Fast spiking interneurons fire high-frequency tonic spikes with relatively constant period. $20\dot{v_m} = (v_m + 55)(v_m + 40) - u + I$, $\dot{u} = 0.15((v_m + 55) - u)$, if $v_m \geq 25$, then $v_m \leftarrow -55$, $u \leftarrow u + 200$.

10 Despite the simplicity of a simulation, analysis of Izhikevich model is difficult since "if" statement is involved (see [15] for more details).



(a) Regular spiking (RS) neuron

(b) Intrinsically bursting (IB) neuron

(c) Chattering (CH) neuron

(d) Fast spiking (FS) neuron

(e) Low-threshold spiking (LTS) neuron

(f) Late spiking (LS) neuron

Figure 2.9: Response of Izhikevich (2014) models to the step synaptic drive. (a) (c) is the excitatory neuron and (d) (f) is the inhibitory neuron. The step synaptic drive of 100 $[pA]$ was applied during $50 \sim 350$ $[ms]$.

E. *Low threshold spiking (LTS) neuron:* (Figure 2.9e) $100\dot{v_m} = (v_m + 56)(v_m + 42) - u + I$, $\dot{u} = 0.03(8(v_m + 56) - u)$, if $v_m \geq 40$, then $v_m \leftarrow -53$, $u \leftarrow u + 20$.

F. *Late spiking (LS) neuron:* (Figure 2.9f) $20\dot{v_m} = 0.3(v_m + 66)(v_m + 40) + 1.2(v_d - v_m) - u + I$, $\dot{u} = 0.17(5(v_m + 66) - u)$, $\dot{v_d} = 0.01(v_m - v_d)$ if $v_m \geq 30$, then $v_m \leftarrow -45$, $u \leftarrow u + 100$.

2.3.4 *Intrinsic Plasticity*

The intensity of an average synaptic input in the brain may change dramatically. Neurons maintain responsiveness to both small and large synaptic inputs by regulating intrinsic excitability to promote stable firing. This way, neuronal activity can keep from falling silent or

saturating when the average synaptic input falls extremely low or rises significantly high. Intrinsic plasticity (IP) regulates the firing rate of a neuron within an appropriate range [96, 153]. The firing rate entropy can be influenced by the neuron's intrinsic properties. By changing these intrinsic properties, the neuron can achieve the optimal firing rate distribution.

Li et al. introduced IP on Izhikevich neuron and showed that Liquid State Machine (LSM) with Spike-Time Dependent Plasticity (STDP) and IP performs better than LSM with a random SNN or SNN obtained by STDP alone. The parameter b in Equation 2.17 governs the excitability of a neuron and IP on Izhikevich neuron can be formulated as follows [96]:

See Section 2.4.3 for more details about STDP

$$\phi = \begin{cases} -\eta \exp(\frac{\tau_{min} - \Delta t_{ISI}}{\tau_{min}}) & \text{if } \Delta t_{ISI} < T_{min} \\ \eta \exp(\frac{\Delta t_{ISI} - \tau_{max}}{\tau_{max}}) & \text{if } \Delta t_{ISI} > T_{max} \\ 0 \end{cases} \tag{2.19}$$

$$b \leftarrow b + b_{max} \cdot \phi \tag{2.20}$$

where η is a learning rate, T_{min} and T_{max} are thresholds that determine the desired range of inter-spike interval (ISI) represented as Δt_{ISI}.

During the training, the most recent ISI is examined and neuronal excitability is adjusted. When ISI is larger than the threshold T_{max}, the neuronal excitability is strengthened to make the neuron more sensitive to input stimuli, and if ISI is less than the threshold T_{min}, the neuronal excitability is weakened to make the neuron less sensitive to input stimuli.

2.4 SYNAPSE MODELS

There are two types of synapses: chemical synapses and electrical synapses of the gap junction. Although both exist in the central nervous system, we will consider only chemical synapses in this article.

2.4.1 *Synaptic Kinetics*

The pre-synaptic spike trains are not transmitted directly to the next neuron due to the spacial gap between the neurons. Synaptic kinetics is defined by the amount of neurotransmitter released from the pre-synaptic cell, the amount of neurotransmitter bonded to the post-synaptic cell, or opening rate of ion channel of the post-synaptic cell. The double exponential model is the one of the simplest model that While ignoring the physiological process, the double exponential model reproduces the behavior of the post-synaptic current (PSC)

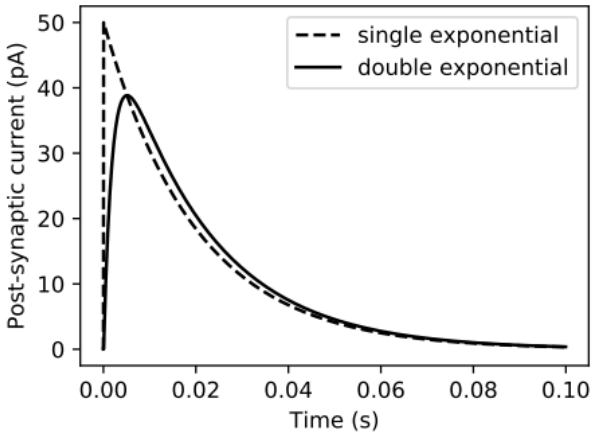


Figure 2.10: Response of synaptic kinetics models overtime. (dot): single exponential model, (line): double exponential model.

well considering not only decay but rising of the PSC. The double exponential model can be expressed in a form of differential equations as follows:

$$\begin{aligned} \frac{ds_{syn}}{dt} &= -\frac{s_{syn}}{\tau_d} + h \\ \frac{dh}{dt} &= -\frac{h}{\tau_r} + \frac{1}{\tau_r \tau_d} \sum \delta(t - t_k) \end{aligned} \tag{2.21}$$

where s_{syn} is synaptic kinetics, τ_r is synaptic rising time constant, τ_d is synaptic decay time constant, and $\delta(\cdot)$ is the Dirac delta function that represents the occurrence of k^{th} spike in a pre-synaptic neuron.

2.4.2 *Synaptic Drive*

Two kinds of synapse model are widely used in computational neuroscience, namely, current- and conductance-based synapses. The difference between current- and conductance-based synapses lied in the definition of these synaptic input currents [24]. In real neural systems, synaptic current induced by an input spike is dependent on the voltage of the post-synaptic neuron.

Abstract Synapse

Disregarding the biological plausibility, synaptic connection can be simply implemented as the product of pre-synaptic spike occurrence and the synaptic weight. This implementation allows to inter-connect neurons with convolutional connection, which has showed great success in computer vision tasks. This is because each synaptic connection

TYPE OF POST NEURON	$\text{AMPA}_{recurrent}$	$\text{AMPA}_{external}$	GABA
excitatory	-10.5	-13.75	42.5
inhibitory	-14	-19	54

Table 2.2: Synaptic efficiencies $J_{syn}\ [pA]$ of the current-based network (adopted from [24])

is independent from neuron's dynamics and therefore the synaptic weight can be shared across the entire receptive field.

$$I_{syn}(t) = \sum_j W_{i,j} \cdot \delta(t) \tag{2.22}$$

where $W_{i,j}$ represents the synaptic weight between the j^{th} pre-neuron and the i^{th} post-neuron, and $\delta(\cdot)$ is the Dirac delta function that represents the occurrence of spike in a pre-synaptic neuron.

Current Based Synapse

Current-based synapse models neglect the voltage-dependent properties of synaptic currents and simply represents the synaptic drive by the change in input current. However, as Wang and Wong suggests, the voltage-dependent properties of synaptic currents play a key role in robust network synchronization in most cases; therefore, current-based synapses are potentially oversimplified for analyzing robust network [162]. In current-based synapse models, synaptic drive I_{syn} can be expressed as a product of synaptic efficiency J and the synaptic kinetics s_{syn}.

$$I_{syn}(t) = -\sum_j J_{i,j} s_{syn}(t) \tag{2.23}$$

where $J_{i,j}$ represents the synaptic efficiency between the j^{th} pre-neuron and the i^{th} post-neuron. The synaptic kinetics s_{syn} can be expressed using Equation 2.21. In this model, the learnable parameter is the synaptic efficiency J.

Conductance Based Synapse

Conductance based synapse represents the synaptic drive by the change in conductance of ion channels. In this model, post- synaptic currents are dependent on the membrane potential of post-synaptic neurons.

$$I_{syn}(t) = -(v_m^{post} - E_{syn}^{+}) g_{i,j}^{+} - (v_m^{post} - E_{syn}^{-}) g_{i,j}^{-} \tag{2.24}$$

where $\pm$ represents the excitatory and inhibitory receptors respectively, E_{syn} is the reversal potential of synapse that depends on the type of

TYPE OF POST NEURON	AMPA$_{recurrent}$	AMPA$_{external}$	GABA
excitatory	0.178	0.234	2.01
inhibitory	0.233	0.317	2.70

Table 2.3: Synaptic conductance G_{syn} $[nS]$ of the conductance-based network (adopted from [24])

the pre-synaptic neuron (typically $E^{+}_{syn} = 0mV$ and $E^{-}_{syn} = -75mV$), and $g_{i,j}$ is the positive value that represents the synaptic conductance between j^{th} pre-neuron and i^{th} post-neuron.

$$g^{\pm}_{i,j} = \sum_{j} G^{\pm}_{i,j} s_{syn}(t) \geq 0 \tag{2.25}$$

In this model, $G \geq 0$ is the learnable parameter that can be trained through synaptic plasticity.

2.4.3 *Synaptic Plasticity*

Synaptic plasticity is the biological process by which specific patterns of synaptic activity result in changes in synaptic strength. Synaptic plasticity was first proposed as a mechanism for learning and memory on the basis of theoretical analysis by Donald Hebb in 1949 [63]. This is often summarized by the phrase "Cells that fire together, wire together."

Spike-Time Dependent Plasticity

Spike-Time Dependent Plasticity (STDP) is an unsupervised Hebbian learning mechanism, which adjusts synaptic weight based on the temporal order of the pre- and post-synaptic spikes [18, 151]. When the pre-synaptic spike arrives before a post-synaptic spike, the synaptic weight is increased, which is known as long term potential (LTP). If the arrival timing of the synaptic spike is reversed, the synaptic weight is decreased, which is known as long-term depression (LTD).

$$\Delta w = \begin{cases} A_{+} \exp(\frac{t_{pre} - t_{post}}{\tau_{+}}) & \text{if } t_{pre} \leq t_{post} \\ -A_{-} \exp(-\frac{t_{pre} - t_{post}}{\tau_{-}}) & \text{if } t_{pre} > t_{post} \end{cases} \tag{2.26}$$

Equation 2.26 suggests that the synaptic strength can be increased or decreased infinitely, which is biologically unrealistic. Biological neurons have a capacity to regulate their own excitability relative to network activity by decreasing the strength of each synapse so that the relative synaptic weighting of each synapse is preserved [149]. This phenomena is called homeostatic scaling and can be implemented

by making $A_{\pm}$ weight dependant such that $w_{min} < w < w_{max}$ is satisfied[11].

$$\begin{cases} A_+(w) = \eta_+(w_{max} - w) \\ A_-(w) = \eta_-(w - w_{min}) \end{cases} \tag{2.27}$$

Here, $\eta_{\pm}$ are learning rates that take small positive values.

In terms of biology as well as implementation, it is infeasible to remember all the time of spike occurrence as seen in Equation 2.26. This is where the spike trace x is introduced.

$$\frac{dw}{dt} = A_+ x_{pre} \cdot \delta_{post} - A_- x_{post} \cdot \delta_{pre} \tag{2.28}$$

$$\begin{aligned} \frac{dx_{pre}}{dt} &= -\frac{x_{pre}(t)}{\tau_+} + \delta(t) \\ \frac{dx_{post}}{dt} &= -\frac{x_{post}(t)}{\tau_-} + \delta(t) \end{aligned} \tag{2.29}$$

where τ_+ and τ_- are the time constants. Figure 2.11 shows the response of a spike trace and corresponding weight modifications based on STDP.

Dopamine Modulated STDP

While STDP operates based upon the correlation between the spike timings of the pre- and post-synaptic neurons, a reward is introduced to modulate STDP in order to implement reinforcement learning mechanism (Section 2.7). If the reward is positive, the corresponding synapse is potentiated, otherwise, the corresponding synapse is depressed. According to Izhikevich, dopaminergic neurons are characterized as having two different firing patterns. In the absence of any stimulus they exhibit a slow (1-8Hz) firing rate, known as background firing. When stimulated the dopaminergic neurons exhibit burst firing. Burst firing is where neurons fire in very rapid bursts, followed by a period of inactivity [74]. The modulation is done by introducing an eligibility trace z for pre- and post-synaptic spike occurrence.

$$\frac{dw}{dt} = \eta r(t) z_{i,j}(t) \tag{2.30}$$

$$\frac{dz_{i,j}}{dt} = -\frac{z_{i,j}(t)}{\tau} + STDP(t) \tag{2.31}$$

11 There are two ways for bounding the weight: soft bound and hard bound

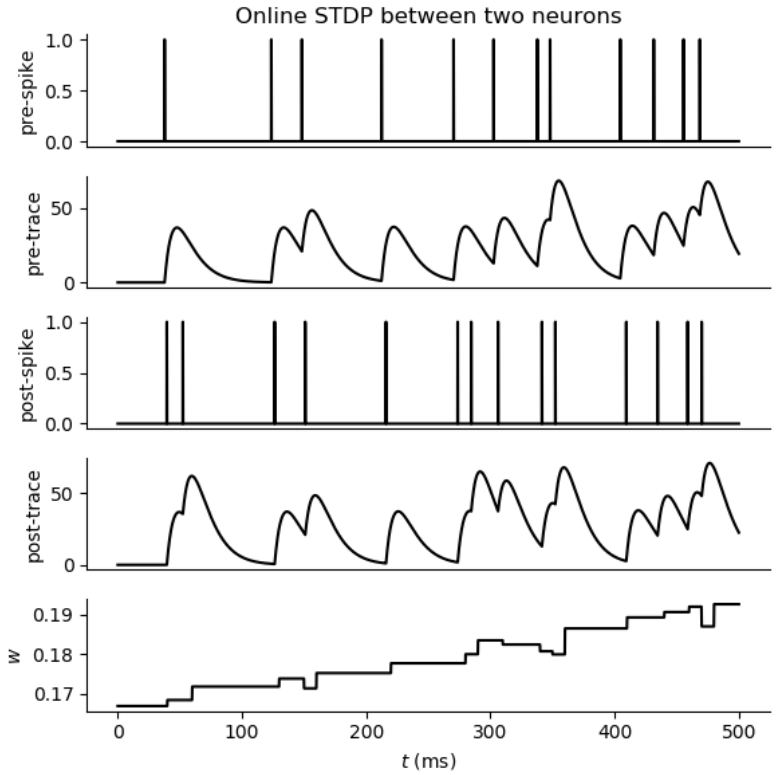


Figure 2.11: Weight change in two neurons based on STDP learning rule.

where $r(t)$ is the reward given at time t.

This plasticity can be used not only for reinforcement learning but for supervised learning. used this plasticity along with STDP to achieve 97.2% on MNIST digit classification with convolutional SNN [20].

Prescribed Error Sensitivity

Prescribed Error Sensitivity (PES) is a supervised learning rule that learns a function by minimizing an external error signal [11]. This rule has been used for many works including biologically detailed neural model of hierarchical reinforcement learning [135] and adoptive control of quadcopter flight [81]. The weight update for this rule is defined as follows:

$$\frac{dw}{dt} = \kappa e(t) a \tag{2.32}$$

where κ is a learning rate, $e(t)$ is a error signal at time t, and a is the rate activity of each neuron.

2.5 OVERVIEW OF DEEP LEARNING

A general feed forward neural network consists of L hidden layers of units, including one layer of input units and one layer of output units. The number of input units is N, output units M, and units in hidden layer l is N^l. The weight of the j^{th} unit in layer l and the i^{th} unit in layer $l+1$ is denoted by w_{ij}^l. The activation of the i^{th} unit in layer l is $\mathbf{h}_i^l$. The input and output of the network are denoted as $\mathbf{x}(n)$, $\mathbf{o}(n)$, respectively, where n denotes training instance, not time.

Two main types of neural networks, i.e. Convolutional Neural Networks and Recurrent Neural Networks are introduced as follows:

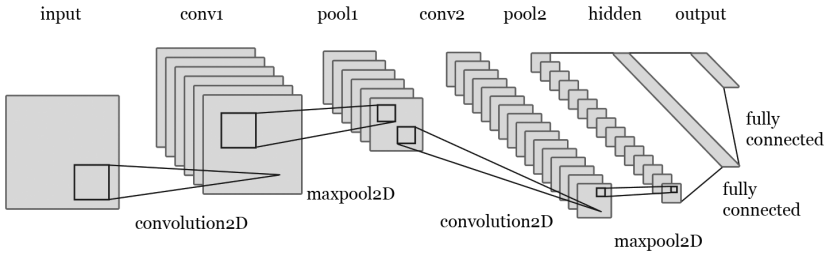


Figure 2.12: Architecture of a typical convolutional network for image classification (LeNet-5) containing three basic layers: convolution layer, pooling layer and fully connected layer

2.5.1 *Convolutional Neural Networks*

Convolutional Neural Networks (CNNs) [88, 89] are a special case of fully connected multi-layer perceptrons that implement weight sharing for processing data that has a known, grid-like topology (e.g. images). CNNs use the spatial correlation of the signal to utilize the architecture in a more sensible way. Their architecture, somewhat inspired by the biological visual system, possesses two key properties that make them extremely useful for image applications: spatially shared weights and spatial pooling. These kind of networks learn features that are shift-invariant, i.e., filters that are useful across the entire image (due to the fact that image statistics are stationary). The pooling layers are responsible for reducing the sensitivity of the output to slight input shift and distortions, and increasing the reception field for later layers. Since 2012, one of the most notable results in Deep Learning is the use of convolutional neural networks to obtain a remarkable improvement in object recognition for ImageNet classification challenge [39, 84].

A typical convolutional network is composed of multiple stages, as shown in Fig. Figure 2.12. The output of each stage is made of a set of 2D arrays called feature maps. Each feature map is the outcome of one convolutional (and an optional pooling) filter applied over the full image. A point-wise non-linear activation function is applied after

each convolution. In its more general form, a convolutional network can be written as:

$$\begin{aligned} \mathbf{h}^0 &= \mathbf{x} \\ \mathbf{h}^l &= pool^l(\sigma_l(\mathbf{w}^l\mathbf{h}^{l-1} + \mathbf{b}^l)), \forall l \in 1, 2, ...L \\ \mathbf{o} &= \mathbf{h}^L \end{aligned} \tag{2.33}$$

where $\mathbf{w}^l, \mathbf{b}^l$ are trainable parameters at layer l. $\mathbf{x} \in \mathrm{R}^{c \times h \times w}$ is vectorized from an input image with c channels, h is the image height and w is the image width. $\mathbf{o} \in \mathrm{R}^{n \times h' \times w'}$ is vectorized from an array of dimension $h' \times w'$ of output vector (of dimension n). $pool^l$ is a (optional) pooling function at layer l.

CNNs have been applied in *image classification* for a long time [43]. Compared to traditional methods, CNNs achieve better classification accuracy on large scale datasets [39] [142]. With large number of classes, proposing a hierarchy of classifiers is a common strategy for image classification [159]. *Visual tracking* is an another application that turns the CNNs model from a detector into a tracker [46]. As a special case of image segmentation, *saliency detection* is another computer vision application that uses CNNs [160], [94]. In additional to the previous applications, *pose estimation* [129], [158] is another interesting research that uses CNNs to estimate human-body pose. *Action recognition* in both still images and in videos are special case of recognition and are challenging problems. [54] utilizes CNN-based representation of contextual information in which the most representative secondary region within a large number of object proposal regions together the contextual features are used to describe the primary region. CNNs-based action recognition in video sequences are reviewed in [174]. *Text detection and recognition* using CNNs is the next step of optical character recognition (OCR) [168] and word spotting [75]. Going beyond still images and videos, *speech recognition and speech synthesis* is also an important research field that have been improved by applying CNNs [26, 169]. In short, CNNs have made breakthroughs in many computer vision areas i.e image, video, speech and text.

2.5.2 *Recurrent Neural Networks (RNNs)*

RNNs is an extremely powerful sequence model and was introduced in the early 1990s [77]. A standard RNNs contains three parts, namely, sequential input data ($\mathbf{x}_t$), hidden state ($\mathbf{h}_t$) and sequential output data ($\mathbf{o}_t$) as shown in Fig. Figure 2.13.

RNNs make use of sequential information and perform the same task for every element of a sequence where the output is dependent on the previous computations. The activation of the hidden states at time-step t is computed as a function f of the current input symbol $\mathbf{x_t}$

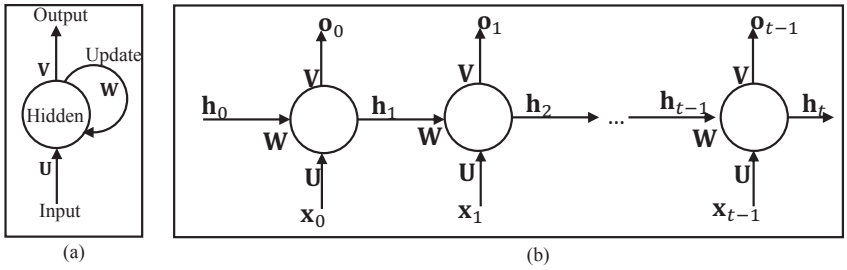


Figure 2.13: An RNNs and the unfolding in time of the computation involved in its forward computation.

and the previous hidden states $\mathbf{h_{t-1}}$. The output at time t is calculated as a function g of the current hidden state $\mathbf{h_t}$ as follows:

$$\begin{aligned} \mathbf{h}_t &= f(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1}) \\ \mathbf{o}_t &= g(\mathbf{V}\mathbf{h}_t) \end{aligned} \tag{2.34}$$

where $\mathbf{U}$ is the input-to-hidden weight matrix, $\mathbf{W}$ is the state-to-state recurrent weight matrix, $\mathbf{V}$ is the hidden-to-output weight matrix. f is usually a logistic sigmoid function or a hyperbolic tangent function and g is defined as a softmax function.

Most work on RNNs has made use of the method of backpropagation through time (BPTT) [140] to train the parameter set $(\mathbf{U}, \mathbf{V}, \mathbf{W})$ and propagate error backward through time. In classic backpropagation, the error or loss function is defined as:

$$E(\mathbf{o}, \mathbf{y}) = \sum_t ||\mathbf{o}_t - \mathbf{y}_t||^2 \tag{2.35}$$

where $\mathbf{o}_t$ is prediction and $\mathbf{y}_t$ is labeled groundtruth.

For a specific weight $\mathbf{W}$, the update rule for gradient descent is defined as $\mathbf{W}^{new} = \mathbf{W} - \gamma \frac{\partial E}{\partial \mathbf{W}}$, where γ is the learning rate. In RNNs model, the gradients of the error with respect to our parameters $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{W}$ are learned using Stochastic Gradient Descent (SGD) and chain rule of differentiation.

The difficulty of training an RNNs to capture long-term dependencies has been studied in [13]. To address the issue of learning long-term dependencies, Hochreiter and Schmidhuber [66] proposed Long Short-Term Memory (LSTM), which is able to maintain a separate memory cell inside it that updates and exposes its content only when deemed necessary. Recently, a Gated Recurrent Unit (GRU) was proposed by [27] to make each recurrent unit adaptively capture dependencies of different time scales. Like the LSTM unit, the GRU has gating units that modulate the flow of information inside the unit, but without having separate memory cells.

Several variants of RNNs have been later introduced and successfully applied to wide variety of tasks, such as natural language processing [95, 108], speech recognition [29, 56], machine translation [78,

102], question answering [65], image captioning [42, 103], and many more.

2.6 TRAINING OF DEEP SPIKING NEURAL NETWORKS

Unlike the DNNs, which can be successfully trained by backpropagation, deep SNNs still do not have solid optimization methods. According to Pfeiffer and Pfeil, five main strategies for training deep SNNs have been developed [131]. The five strategies include: (1) binarization of ANNs, (2) model conversion from ANNs, (3) training of constrained ANNs, (4) supervised training with spike, and (5) local learning rule at synapse. Among those methods, state-of-the-art result is obtained from the model conversion from ANN.

2.7 REINFORCEMENT LEARNING

This section serves as a brief introduction to the theoretical models and techniques in reinforcement learning (RL). In order to provide a quick overview of what constitutes the main components of RL methods, some fundamental concepts and major theoretical problems are also clarified. RL is a kind of machine learning methods where agents learn the optimal policy by trial and error. Unlike supervised learning, the feedback is available after each system action, it is simply a scalar value which may be delayed in time in RL framework, for example the success or failure of the entire system is reflected after a sequence of actions. Furthermore, supervised learning model is updated based on the loss/error of the output and there is no mechanism to get correct value when it is wrong. This is addressed by policy gradients in RL by assigning gradients without a differentiable loss function which aims at teaching a model to try things out randomly and learn to do correct things more, and keep error less.

Stimulated by behavioral psychology, RL was proposed to address the sequential decision-making problems which are commonly existed in many applications such as game, robotics, healthcare, smart grids, stock, autonomous driving, etc. Unlike the supervised learning where the data is given, an artificial agent collects experiences (data) by interacting with its environment in RL framework. Such experience is then gathered to optimize the cumulative rewards/utilities.

In this section, we focus on how the RL problem can be formalized as an agent that is able to make decisions in an environment to optimize some objectives presented under reward functions. Some key aspects of RL are: (i) Address the sequential decision making; (ii) There is no supervisor, only a reward presented as scalar number; and (iii) The feedback is highly delayed. Markov Decision Process (MDP) is a framework that has commonly been used to solve most RL problems with discrete actions, thus we will firstly discuss about MDP in this

section. We then introduce value function and how to categorize RL into model-based or model-free methods. At the end of this section, we discuss some challenges in RL.

The standard theory of reinforcement learning is defined for a Markov Decision Process (MDP), as shown in Figure 2.14. MDP is a mathematical framework for modeling discrete-time stochastic control process defined by a 4-tuple (state s, action a, transition probability $T(s_{t+1}|s_t, a_t)$, reward $r(s_t, s_{t+1})$).

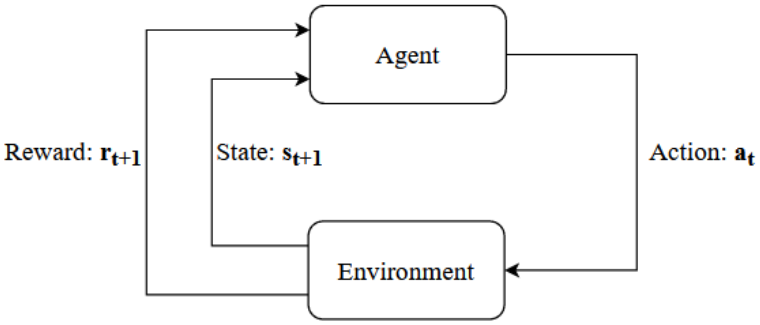


Figure 2.14: Agent-Environment interaction in reinforcement learning

Considering MDP, the agent chooses an action a_t according to the policy $\pi(a_t|s_t)$ at state s_t. Notably, agent's algorithm for choosing action a given current state s, which in general can be viewed as distribution $\pi(a|s)$, is called a policy (strategy). The environment receives the action, produces a reward r_{t+1} and transfers to the next state s_{t+1} according to the transition probability $T(s_{t+1}|s_t, a_t)$. The process continues until the agent reaches a terminal state or a maximum time step. In RL frameworks, the tuple $(s_t, a_t, r_{t+1}, s_{t+1})$ is called transition. Several sequential transitions are usually referred to as roll-out. Full sequence $(s_0, a_0, r_1, s_1, a_1, r_2, ...)$ is called a trajectory. Theoretically, trajectory is infinitely long, but the episodic property holds in most practical cases. One trajectory of some finite length τ, is called an episode. For given MDP and policy π, the probability of observing $(s_0, a_0, r_1, s_1, a_1, r_2, ...)$ is called *trajectory distribution* and is denoted as:

$$\mathcal{T}_\pi = \prod_t \pi(a_t|s_t) T(s_{t+1}|s_t, a_t) \tag{2.36}$$

The objective of reinforcement learning is to find the *optimal policy* π^* for the agent that maximizes the cumulative reward, called *return*. For every episode, return is defined as the weighted sum of immediate rewards:

$$\mathcal{R} = \sum_{t=0}^{\tau-1} \gamma^t r_{t+1} \tag{2.37}$$

Because the policy induces a trajectory distribution, the *expected reward* maximization can be written as:

$$\mathbb{E}_{\mathcal{T}_\pi} \sum_{t=0}^{\tau-1} r_{t+1} \rightarrow \max_\pi \tag{2.38}$$

Thus, given MDP and policy π, the *discounted expected reward* is defined:

$$\mathcal{G}(\pi) = \mathbb{E}_{\mathcal{T}_\pi} \sum_{t=0}^{\tau-1} \gamma^t r_{t+1} \tag{2.39}$$

The goal of RL is to find an *optimal policy* π^*, which maximizes the discounted expected reward, i.e. $\mathcal{G}(\pi) \rightarrow \max_\pi$

2.7.1 *Value functions*

In order to estimate how good it is for an agent to utilize policy π to visit state s, a value function is introduced. The value is the mathematical expectation of return and value approximation is obtained by Bellman expectation equation as follows:

$$V^\pi(s_t) = \mathbb{E}[r_{t+1} + \gamma V^\pi(s_{t+1})] \tag{2.40}$$

$V^\pi(s_t)$ is also known as state-value function, and the expectation term can be expanded as a product of policy, transition probability, and return as follows:

$$V^\pi(s_t) = \sum_{a_t \in \mathcal{A}} \pi(a_t|s_t) \sum_{s_{t+1} \in S} T(s_{t+1}|s_t, a_t)[R(s_t, s_{t+1}) + \gamma V^\pi(s_{t+1})] \tag{2.41}$$

This equation is called Bellman equation. When the agent always selects the action according to the optimal policy π^* that maximizes the value, Bellman equation can be expressed as following:

$$\begin{aligned} V^*(s_t) &= \max_{a_t} \sum_{s_{t+1} \in S} T(s_{t+1}|s_t, a_t)[R(s_t, s_{t+1}) + \gamma V^*(s_{t+1})] \\ &\triangleq \max_{a_t} Q^*(s_t, a_t) \end{aligned} \tag{2.42}$$

However, obtaining optimal value function V^* doesn't provide enough information to reconstruct some optimal policy π^* because of the complexity of the real world. Thus, a quality function (Q-function) under policy π is introduced as:

$$Q^\pi(s_t, a_t) = \sum_{s_{t+1}} T(s_{t+1}|s_t, a_t)[R(s_t, s_{t+1}) + \gamma V^\pi(s_{t+1})] \tag{2.43}$$

Unlike value function which specifies the goodness of a state, a Q-function specifies the goodness of an action in a state.

Table 2.4: Comparison between model-based RL and model-free RL

Factors	**Model-based RL**	**Model-free RL**
Number of iterations between agent and environment	Small	Big
Convergence	Fast	Slow
Prior knowledge of transitions	Yes	No
Flexibility	Strongly depend on a learnt model	Adjust based on trials and errors

2.7.2 *Category*

In general, RL can be divided into either model-free or model-based methods. Here, "model" is defined by the two quantity: transition probability function $T(s_{t+1}|s_t, a_t)$ and the reward function $R(s_t, s_{t+1})$. The comparison between model-based and model-free methods is given in Table 2.4.

Model-based RL

Model-based RL is an approach that uses a learnt model, i.e. $T(s_{t+1}|s_t, a_t)$ and reward function $R(s_t, s_{t+1})$ to predict the future action. There are four main model-based techniques as follows:

- **Value Function:** The objective of value function methods is to obtain the best policy by maximizing the value functions in each state. A value function of a RL problem can be defined as in Equation 2.41 and the optimal state-value function is given in Equation 2.42 which are known as Bellman equations. Some common approaches in this group are Differential Dynamic Programming [93, 115], Temporal Difference Learning [104], Policy Iteration [146] and Monte Carlo [64].

- **Transition Models:** Transition models decide how to map from a state s, taking action a to the next state (s') and it strongly affects the performance of model-based RL algorithms. Based on whether predicting the future state s' is based on probability distribution of a random variable or not, there are two main approaches in this group: stochastic and deterministic. Some common methods for deterministic models are decision trees [122] and linear regression [114]. Some common methods for stochastic models are Gaussian processes [5, 38, 112], Expectation-Maximization [31] and dynamic Bayesian networks [122].

- **Policy Search:** Policy search approach directly searches for the optimal policy by modifying its parameters, whereas the value

function methods indirectly find the actions that maximize the value function at each state. Some of the popular approaches in this group are: gradient-based [44, 116], information theory [85, 112], and sampling based [9].

- **Return Functions:** Return functions decides how to aggregates rewards or punishments over an episode. It affects both the convergence and the feasibility of the model. There are two main approaches in this group: discounted returns functions [9, 40, 165], and averaged returns functions [1, 21]. Between two approaches, the former is the most popular which represents the uncertainty about future rewards. While small discount factors provide a faster convergence, its solution may not be optimal.

In practice, transition and reward functions are rarely known and hard to model. The comparative performance among all model-based techniques is reported in [161] with over 18 benchmarking environments including noisy environments. The Fig.2.16 summarizes different model-based RL approaches.

Model-free methods

Learning through the experience gained from interactions with the environment, i.e. model-free method tries to estimate the t. discrete problemsransition probability function and the reward function from the experience to exploit them in acquisition of policy. Policy gradient and value-based algorithms are popularly used in model-free methods.

- **The policy gradient methods**: In this approach, RL task is considered as an optimization with stochastic first-order optimization. Policy gradient methods directly optimize the discounted expected reward, i.e. $\mathcal{G}(\pi) \rightarrow \max_{\pi}$ to obtains the optimal policy π^* without any additional information about MDP. To do so, approximate estimations of gradient with respect to policy parameters are used. Take [164] as an example, policy gradient parameterizes the policy and updates parameters θ,

$$\mathcal{G}^{\theta}(\pi) = \mathbb{E}_{\mathcal{T}_{\phi}} \sum_{t=0} log(\pi_{\theta}(a_t|s_t))\gamma^t \mathcal{R} \tag{2.44}$$

  where $\mathcal{R}$ is the total accumulated return and defined in Equation 2.37. Common used policies are Gibbs policies [10] and Gaussian policies [130]. Gibbs policies is used in discrete problems whereas Gaussian policies is used in continuous problems.

- **Value-based methods**: In this approach, the optimal policy π^* is implicitly conducted by gaining an approximation of optimal Q-function $Q^*(s, a)$. In value-based methods, agents update the value function to learn suitable policy while policy-based RL agents learn the policy directly. To do that, Q-learning is a typical

value-based method. The update rule of Q-learning with learning rate λ is defined as:

$$Q(s_t, a_t) = Q(s_t, a_t) + \lambda\delta_t \tag{2.45}$$

where $\delta_t = R(s_t, s_{t+1}) + \gamma \arg\max_a Q(s_{t+1}, a) - Q(s_t, a)$ is the temporal difference (TD) error.

Target at self-play Chess, [166] investigates inasmuch it is possible to leverage the qualitative feedback for learning an evaluation function for the game. [141] provides the comparison of learning of linear evaluation functions between using preference learning and using least squares temporal difference learning, from samples of game trajectories. The value-based methods depend on a specific, optimal policy, thus it is hard for transfer learning.

- **Actor-critic** is an improvement of policy gradient with an value-based critic Γ, thus, Equation 2.44 is rewritten as:

$$\mathcal{G}^\theta(\pi) = \mathbb{E}_{\mathcal{T}_\phi} \sum_{t=0} log(\pi_\theta(a_t|s_t))\gamma^t\Gamma_t \tag{2.46}$$

The critic function Γ can be defined as $Q^\pi(s_t, a_t)$ or $Q^\pi(s_t, a_t) - V_t^\pi$ or $R[s_{t-1}, s_t] + V_{t+1}^\pi - V_t^\pi$

Actor-critic methods are combinations of actor-only methods and critic-only methods. Thus, actor-critic methods have been commonly used RL. Depending on a reward setting, there are two groups of actor-critic methods, namely discounted return [16, 123] and average return [17, 127].

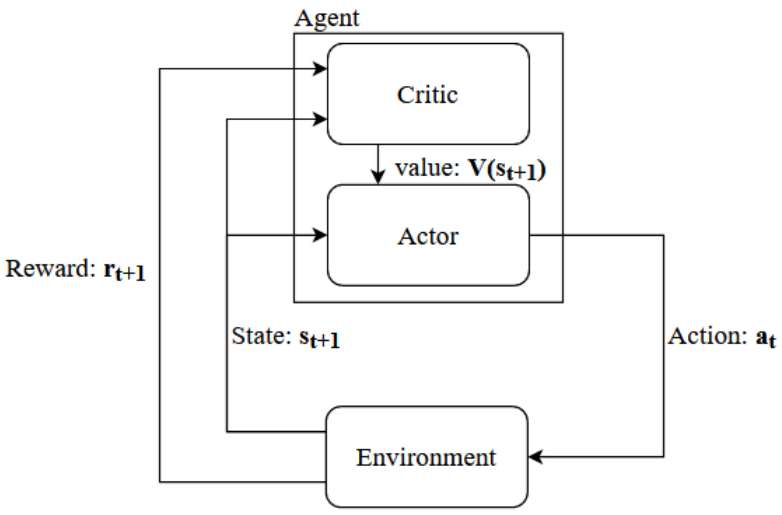


Figure 2.15: Flowchart showing the structure of actor critic algorithm.

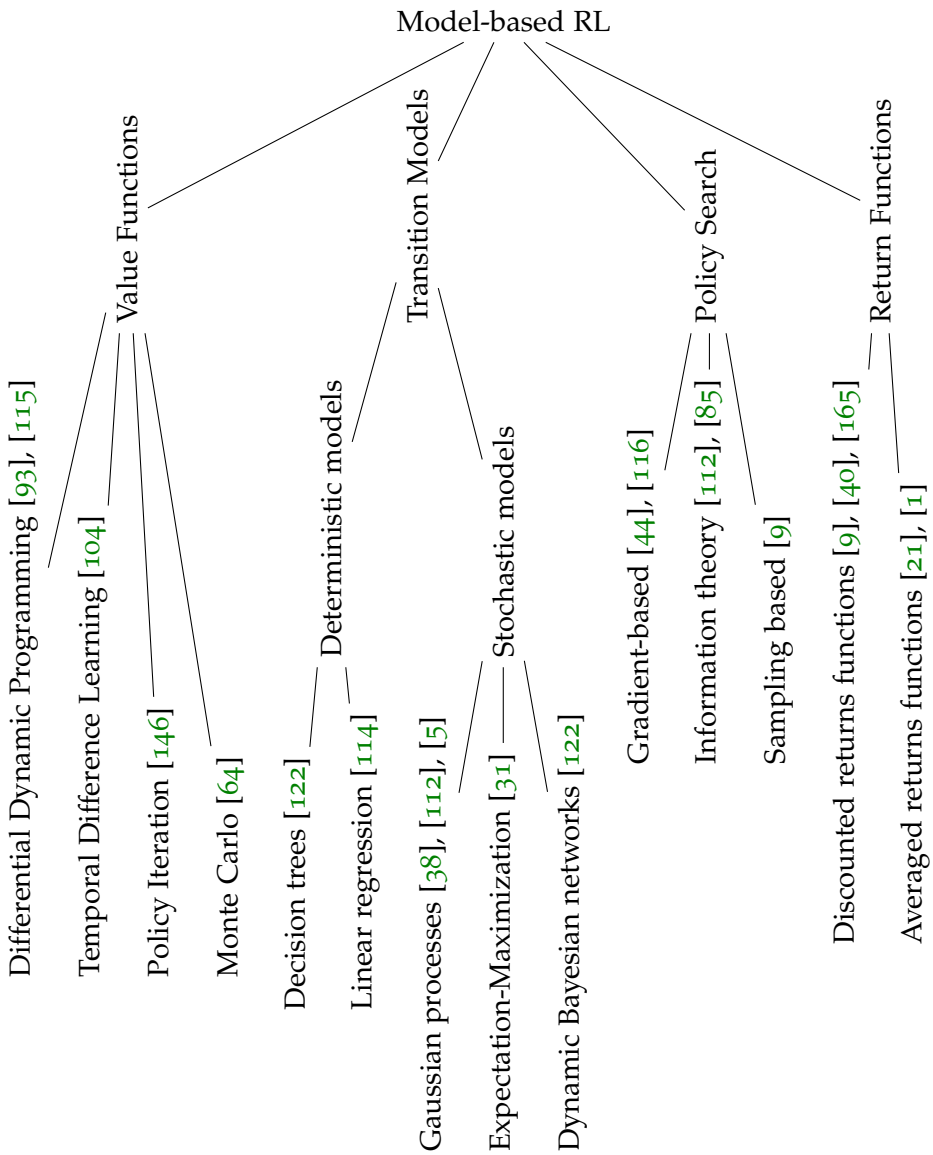


Figure 2.16: Summarization of model-based RL approaches

2.8 DEEP REINFORCEMENT LEARNING

Deep Reinforcement Learning (DRL), which was proposed as a combination of RL and DL, has achieved rapid developments, thanks to the rich context representation of DL. Under DRL, the aforementioned value and policy can be expressed by a neural networks which allows to deal with a continuous state or action that was previously hard for a table representation. Similar to RL, DRL can be categorized into model-based algorithms and model-free algorithms which will be introduced in this section.

2.8.1 *Model-Free Algorithms*

There are two approaches, namely, Value-based DRL methods and Policy gradient DRL methods to implement model-free algorithms.

Value-based DRL methods

Deep Q-Learning Network: Deep Q-Learning [110] (DQN) is the most famous DRL model which learns policies directly from high-dimensional inputs by a deep neural network. In DQN, input is raw pixels and output is value function to estimate future rewards as given in Fig.2.17. Take regression problem as an instance. Let y denote the target of our regression task, the regression with input (s, a), target $y(s, a)$ and the MSE loss function is as:

$$\begin{aligned} \mathcal{L}^{\mathcal{DQN}} = \mathcal{L}(y(s_t, a_t), Q^*(s_t, a_t, \theta_t)) = ||y(s_t, a_t) - Q^*(s_t, a_t, \theta_t)||^2 \\ y(s_t, a_t) = R(s_t, s_{t+1}) + \gamma \max_{a_{t+1}} Q^*(s_{t_1}, a_{t+1}, \theta_t) \end{aligned} \tag{2.47}$$

where θ is vector of parameters, $\theta \in \mathbb{R}^{S \times R}$ and s_{t+1} is a sample from $T(s_{t+1}|s_t, a_t)$ with input of (s_t, a_t). Minimizing the loss function yields a gradient descent step formula to update θ as follows:

$$\theta_{t+1} = \theta_t - \alpha_t \frac{\partial \mathcal{L}^{\mathcal{DQN}}}{\partial \theta} \tag{2.48}$$

Double DQN: In DQN, the values of Q^* in many domains was leading to overestimation because of *max*. In Equation 2.47, $y(s, a) = R(s, s') + \gamma \max_{a'} Q^*(s', a', \theta)$ shifts Q-value estimation towards either to the actions with high reward or to the actions with overestimating approximation error. Double DQN [177] is an improvement of DQN that combines double Q-learning [60] with DQN and it aims at reducing observed overestimation with better performance. The idea of Double DQN is based on separating action selection and action evaluation using its own approximation of Q^* as follows:

$$\max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}; \theta) = Q^*(s_{t+1}, \arg\max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}; \theta_1); \theta_2) \tag{2.49}$$

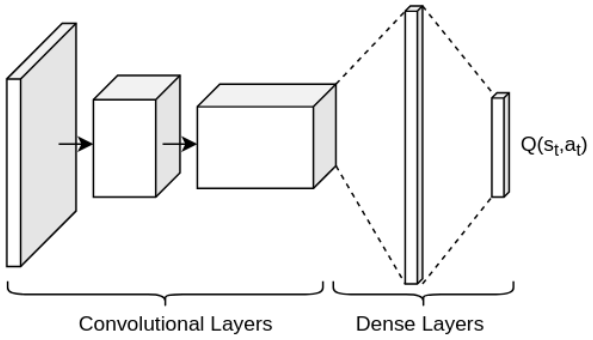


Figure 2.17: Network structure of Deep Q-Network (DQN), where Q-values Q(s,a) are generated for all actions for a given state.

Thus

$$y = R(s_t, s_{t+1}) + \gamma Q^*(s_{t+1}, \arg\max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}; \theta_1); \theta_2) \quad (2.50)$$

The easiest and most expensive implementation of double DQN is to run two independent DQNs as follows:

$$\begin{aligned} y_1 &= R(s_t, s_{t+1}) + \gamma Q_1^*(s_{t+1}, \arg\max_{a_{t+1}} Q_2^*(s_{t+1}, a_{t+1}; \theta_2); \theta_1) \\ y_2 &= R(s_t, s_{t+1}) + \gamma Q_2^*(s_{t+1}, \arg\max_{a_{t+1}} Q_1^*(s_{t+1}, a_{t+1}; \theta_1); \theta_2) \end{aligned} \quad (2.51)$$

Dueling DQN: In DQN, when the agent visits unfavourable state instead of lowering its value V^*, it remembers only low pay-off by updating Q^*. In order to address this limitation, Dueling DQN [163] incorporates approximation of V^* explicitly in computational graph by introducing an advantage function as follows:

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \quad (2.52)$$

Therefore, we can reformulate Q-value: $Q^*(s, a) = A^*(s, a) + V^*(s)$. This implies that after DL the feature map is decomposed into two parts corresponding to $V^*(v)$ and $A^*(s, a)$ as illustrated in Fig.2.18. This can be implemented by splitting the fully connected layers in the DQN architecture to compute the advantage and state value functions separately, then combining them back into a single Q-function. An interesting result has shown that Dueling DQN obtains better performance if it is formulated as:

$$Q^*(s_t, a_t) = V^*(s_t) + A^*(s_t, a_t) - \max_{a_{t+1}} A^*(s_t, a_{t+1}) \quad (2.53)$$

In practical implementation, averaging instead of maximum is used, i.e.

$$Q^*(s_t, a_t) = V^*(s_t) + A^*(s_t, a_t) - \text{mean}_{a_{t+1}} A^*(s_t, a_{t+1})$$

Furthermore, to address the limitation of memory and imperfect information at each decision point, Deep Recurrent Q-Network (DRQN) [61] employed RNNs into DQN by replacing the first fully-connected layer with a RNN. Multi-step DQN [37] is one of the most popular improvement of DQN by substituting one-step approximation with N-steps.

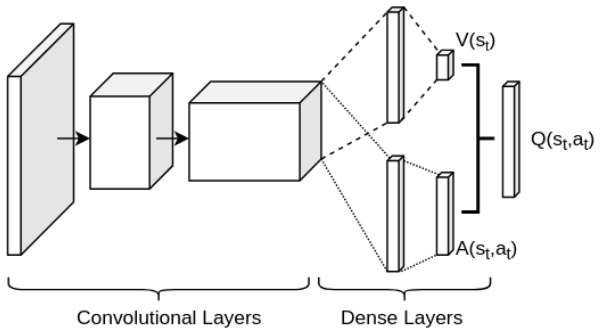


Figure 2.18: Network structure of Dueling DQN, where value function $V(s)$ and advantage function $A(s,a)$ are combined to predict Q-values $Q(s,a)$ for all actions for a given state.

Policy gradient DRL methods

Policy Gradient Theorem: Different from value-based DRL methods, policy gradient DRL optimizes the policy directly by optimizing the following objective function which is defined as a function of θ

$$\mathcal{G}(\theta) = \mathbb{E}_{T\sim\pi_\theta} \sum_{t=1} \gamma^{t-1} R(s_{t-1}, s_t) \rightarrow \max_\theta \tag{2.54}$$

For any MDP and differentiable policy π_θ, the gradient of objective Equation 2.54 is defined by policy gradient theorem [156] as follows:

$$\bigtriangledown_\theta \mathcal{G}(\theta) = \mathbb{E}_{T\sim\pi_\theta} \sum_{t=0} \gamma^t Q^\pi(s_t, a_t) \bigtriangledown_\theta \log \pi_\theta(a_t|s_t) \tag{2.55}$$

REINFORCE: Williams introduced REINFORCE to approximately calculate the gradient in Equation 2.55 by using Monte-Carlo estimation [164]. In REINFORCE approximate estimator, Equation 2.55 is reformulated as:

$$\bigtriangledown_\theta \mathcal{G}(\theta) \approx \sum_{\mathcal{T}}^{N} \sum_{t=0} \gamma^t \bigtriangledown_\theta \log \pi_\theta(a_t|s_t) (\sum_{t'=t} \gamma^{t'-t} R(s_{t'}, s_{t'+1})) \tag{2.56}$$

where $\mathcal{T}$ is trajectory distribution and defined in Equation 2.36. Theoretically, REINFORCE can be straightforwardly applied into any parametric $\pi_\theta(a|s)$. However, it is impractical to use it because of time consuming for convergence and local optimums problem. Based on the observation that convergence rate of stochastic gradient descent directly depends on the variance of gradient estimation, variance reduce

technique was proposed to address naive REINFORCE's limitations by adding a term that reduce the variance without affect the expectation.
Trust Region Policy Optimization (TRPO): TRPO [144] introduces Kullback-Leibler (KL) divergence constraint on the size of the policy update in order to improve the training stability. Specifically, TRPO aims to maximize the weighted advantage under trust region constraint, which enforces the distance between old and new policy distributions measured by KL divergence to be small enough. This constraint can be incorporated within the objective function as follows:

$$\mathbb{E}[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A(s_t,a_t) - \beta KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]] \rightarrow \max_\theta \quad (2.57)$$

Proximal Policy Optimization (PPO): PPO [145] implements the idea of limiting the size of the policy update in a simpler form than TRPO. When the probability ratio between old and new policies is denoted as $r(\theta)$, it needs to stay within a small interval around 1 to prevent sudden change in policy.

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \quad (2.58)$$

PPO realizes this constrain by clipping the ratio to be $1 \pm \epsilon$ and taking the smaller value between the original $r(\theta)$ and the clipped one. This discourages to increase the policy update to extremes for better rewards because this penalizes only the merit from the bigger $r(\theta)$.

$$\mathbb{E}[\min(r(\theta)A(s_t,a_t), clip(r(\theta), 1-\epsilon, 1+\epsilon)A(s_t,a_t))] \quad (2.59)$$

Actor-Critic DRL algorithm

Both value-based and policy gradient algorithms has its own pros and cons, i.e. policy gradient methods are better for continuous and stochastic environments, and have a faster convergence whereas value-based methods are more sample efficient and steady. Lately, actor-critic [82, 111] was proposed to take advantage from both value-based and policy gradient while limiting their drawbacks. Actor-critic architecture computes the policy gradient using a value-based critic function to estimate expected future reward. The principal idea of actor-critic is to divide the model in two parts: (i) computing an action based on a state and (ii) producing the Q values of the action. As given in Fig.2.15, the actor takes as input the state s_t and outputs the best action a_t. It essentially controls how the agent behaves by learning the optimal policy (policy-based). The critic, on the other hand, evaluates the action by computing the value function (value based). The most basic actor-critic method is naive policy gradients (REINFORCE).
Advantage Actor-Critic (A2C): A2C [111] consist of two neural networks i.e. actor network $\pi_\theta(a_t|s_t)$ representing for policy and critic

network V_ω^π with parameters ω approximately estimating actor's performance. In order to determine how much better it is to take a specific action compared to the average, an advantage value is defined as:

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \tag{2.60}$$

Instead of constructing two neural networks for both the Q value and the V value, using Bellman optimization equation we can rewrite the advantage function as:

$$A^\pi(s_t, a_t) = R(s_t, s_{t+1}) + \gamma V_\omega^\pi(s_{t+1}) - V_\omega^\pi(s_t) \tag{2.61}$$

For given policy π, its value function can be obtained using point iteration for solving:

$$V^\pi(s_t) = \mathbb{E}_{a_t \sim \pi(a_t|s_t)} \mathbb{E}_{s_{t+1} \sim T(s_{t+1}|a_t, s_t)} (R(s_t, s_{t+1}) + \gamma V^\pi(s_{t+1})) \tag{2.62}$$

Similar to DQN, on each update a target is computed using current approximation:

$$y = R(s_t, s_{t+1}) + \gamma V_\omega^\pi(s_{t+1}) \tag{2.63}$$

At time step t, the A2C algorithm can be implemented as following steps

- Step 1: Compute advantage function using Equation 2.61.
- Step 2: Compute target using Equation 2.63.
- Step 3: Compute critic loss with MSE loss: $\mathcal{L} = \frac{1}{B} \sum_T ||y - V^\pi(s_t))||^2$, where B is batch size and $V^\pi(s_t)$ is defined in Equation 2.62.
- Step 4: Compute critic gradient: $\bigtriangledown^{critic} = \frac{\partial \mathcal{L}}{\partial \omega}$.
- Step 5: Compute actor gradient: $\bigtriangledown^{actor} = \frac{1}{B} \sum_T \bigtriangledown_\theta \log \pi(a_t|s_t) A^\pi(s_t, a_t)$

Asynchronous Advantage Actor Critic (A3C) Beside A2C, there is another strategy to implement an Actor Critic agent. Asynchronous Advantage Actor Critic (A3C) [111] approach does not use experience replay because this requires lot of memory. Instead, A3C asynchronously execute different agents in parallel on multiple instances of the environment. Each worker (copy of the network) will update the global network asynchronously. Because of the asynchronous nature of A3C, some workers (copy of the agents) will work with older values of the parameters. Thus the aggregating update will not be optimal. On the other hand, A2C synchronously update the global network. A2C waits until all workers finished their training and calculated their gradients to average them, to update the global network. In order to update entire network, A2C waits for each actor to finish their

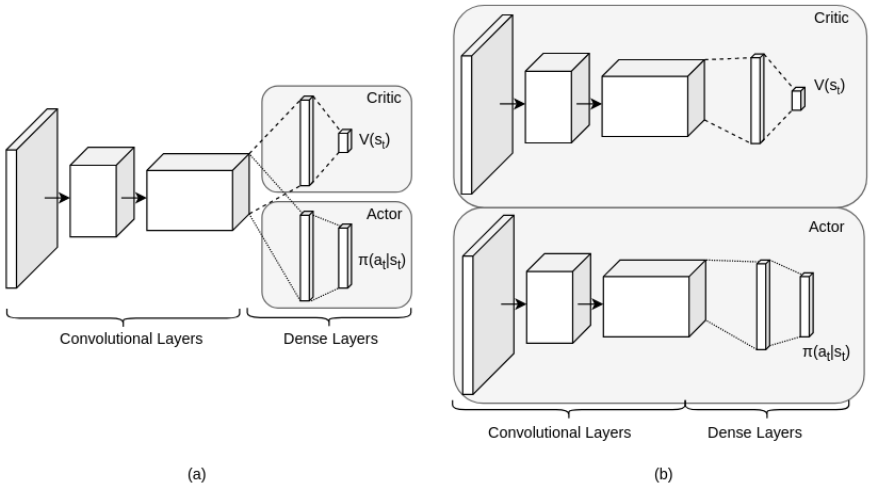


Figure 2.19: An illustration of Actor-Critic algorithm in two cases: sharing parameters (a) and not sharing parameters (b).

segment of experience before updating the global parameters. As a consequence, the training will be more cohesive and faster. Different from A3C, each worker in A2C has the same set of weights since and A2C updates all their workers at the same time. In short, A2C is an alternative of synchronous version of the A3C. In A2C, it waits for each actor to finish its segment of experience before updating, averaging over all of the actors. In practical experiment, this implementation is more effectively uses GPUs due to larger batch sizes. The structure of an actor-critic algorithm can be divided into two types depending on whether or not parameter sharing as illustrated in Fig.2.19.

In order to overcome the limitation of speed, GA3C [8] was proposed and it achieved a significant speed up compared to the original CPU implementation. To more effectively train A3C, [68] proposed FFE which forces on random exploration at the right time during a training episode, that can lead to improved training performance.

2.8.2 *Model-Based Algorithms*

We have discussed so far model-free methods including the value-based approach and policy gradient approach. In this section, we focus on the model-based approach, that deals with the dynamics of the environment by learning a transition model that allows for simulation of the environment without interacting with the environment directly. In contrast to model-free approaches, model-based approaches are learned from experience by a function approximation. Theoretically, no specific prior knowledge is required in model-based RL/DRL but incorporating prior knowledge can help faster convergence and better trained model, speed up training time as well as the amount of training samples. While using raw data with pixel, it is difficult for

model-based RL to work on high dimensional and dynamic environment. This is addressed in DRL by embedding the high-dimensional observations into a lower-dimensional space using autoencoders [48]. Many DRL approaches have been based on scaling up prior work in RL to high-dimensional problems. A good overview of model-based RL for high-dimensional problems, can be found in [132] which partition model-based DRL into three categories, i.e. explicit planning on given transitions, explicit planning on learned transitions, and end-to-end learning of both planning and transitions. In general, DRL targets at training DNNs to approximate the optimal policy π^* together with optimal value functions V^* and Q^*. In the following, we will cover the most common model-based DRL approaches including value function and policy search methods.

Monte Carlo tree search (MCTS): MCTS [32] is one of the most popular method to look-ahead search and it is combined with DNN-based transition model to build a model-based DRL in [3]. In this work, the learned transition model predicts the next frame and the rewards one step ahead using the input of the last four frames of the agent's first-person-view image and the current action. This model is then used by Monte Carlo tree search algorithm to plan the best sequence of actions for the agent to perform.

Value-Targeted Regression (UCRL-VTR): Alex, et al. proposed model-based DRL for regret minimization [76]. In their work, a set of models, that are 'consistent' with the data collected, is constructed at each episode. The consistency is defined as the total squared error, whereas the value function is determined by solving the optimistic planning problem with the constructed set of models

Policy search

Policy search methods aim to directly find policies by means of gradient-free or gradient-based methods.

Model-Ensemble Trust-Region Policy Optimization (ME-TRPO): ME-TRPO [86] is mainly based on Trust Region Policy Optimization (TRPO) [144] which imposes a trust region constraint on the policy to further stabilize learning.

Model-Based Meta- Policy-Optimization (MB-MPO): MB-MPO [30] addresses the performance limitation of model-based DRL compared against model-free DRL when learning dynamics models. MB-MPO learns an ensemble of dynamics models, a policy that can quickly adapt to any model in the ensemble with one policy gradient step. In results, the learned policy exhibits less model-bias without the need to behave conservatively.

A summary of both model-based and model-free DRL algorithms is given in Table Table 2.5.

Table 2.5: Summary of model-based and model-free DRL algorithms consisting of value-based and policy gradient methods.

DRL Algorithms	Description	Category
DQN [110]	Deep Q Network	Value-based, Off-policy
Double DQN [177]	Double Deep Q Network	Value-based, Off-policy
Dueling DQN [163]	Dueling Deep Q Network	Value-based, Off-policy
MCTS [3]	Monte Carlo tree search	Valued-based, On-Policy
UCRL-VTR[76]	optimistic planning problem	Valued-based, On-Policy
DDPG [97]	DQN with Deterministic Policy Gradient	Policy gradient, Off-policy
TRPO [144]	Trust Region Policy Optimization	Policy gradient, On-policy
PPO [145]	Proximal Policy Optimization	Policy gradient, On-policy
ME-TRPO [86]	Model-Ensemble Trust-Region Policy Optimization	Policy gradient, On-policy
MB-MPO [30]	Model-Based Meta- Policy-Optimization	Policy gradient, On-policy
A3C [111]	Asynchronous Advantage Actor Critic	Actor Critic, On-Policy
A2C [111]	Advantage Actor Critic	Actor Critic, On-Policy

2.8.3 *Good practices*

Inspired by Deep Q-learning [110], we discuss some useful techniques that are used during training an agent in DRL framework in practices.

Experience replay

Experience replay [173] is a useful part of off-policy learning and it often used while training an agent in RL framework. By getting rid of as much information as possible from the past experiences, it removes the correlations in training data and reduces the oscillation of learning procedure. In result, it enables agents to remember and re-use the past experiences sometimes in many weights updates which increases data efficiency.

Minibatch learning

Minibatch learning is a common technique that is used together with experience replay. Minibatch allows learning more than one training sample at each step, thus, it makes the learning process robust to outliers and noise.

Target Q-network freezing

As described in [110], two networks are used for training process. In target Q-network freezing: one network interacts with the environment and another network plays a role of target network. The first network is used to generate target Q-values that are used to calculate losses. The weights of the second network i.e. target network are fixed and slowly updated to the first network [98].

Reward clipping

Reward is the scalar number provided by the environment and it aims at optimizing the network. To keep the rewards in a reasonable scale and to ensure proper learning, they are clipped to a specific range (-1 ,1). Here 1 refers to as positive reinforcement or reward and -1 is referred to as negative reinforcement or punishment.

Model-based v.s. Model-free approach

Whether the model-free or model-based approaches is chosen mainly depends on the model architecture i.e. policy and value function.

2.9 ONLINE LEARNING

Online learning exploits a constant stream of data where only a single sample is provided to the learning algorithm at every time step, which

is incrementally updated every time a new sample arrives. ANNs are typically trained with backpropagation in a batch-mode methods, minibatches, and multiple passes over the data, which requires the entire training data to be made available prior to the learning task [143]. Online Learning can be directly applied to DNNs but they suffer from convergence issues, such as vanishing gradients and diminishing feature reuse. There have been attempts on utilization of deep learning with online learning [90, 175]. However, they operate in a sliding window approach with a mini-batch training, making them unsuitable for a streaming data. Regarding the online learning, SNNs are known for their ability to lean continuously and incrementally, which allows them to continuously adapt to non-stationary and evolving environments [100].

2.10 AVAILABLE SOFTWARE FRAMEWORKS

We provide a list of known software frameworks for the SNN simulation with some emphasis on relation with deep learning frameworks in Table 2.6.

2.11 RELATED WORKS

2.11.1 *Deep Learning-based Image Segmentation*

Semantic segmentation refers to associating one of the classes to each pixel in an image. This problem has played an important role in many research areas and has attracted numerous studies recently when deep learning have become ubiquitous in semantic segmentation.

One of the first works on CNN-based semantic segmentation approaches is proposed by Farabet et al. It uses a multiscale convolutional network trained from raw pixels to extract dense feature vectors that encode regions of multiple sizes centered on each pixel [47]. This work consists of two components: multi-scale, convolutional representation and graph-based classification. To simultaneously detect and segment all instances of a category in an image, [59] proposes SDS (Simultaneous Detection and Segmentation) approach which uses CNN to classify region proposals. Later, [101] proposes "fully convolutional" networks that take input of arbitrary size and produce correspondingly-sized output with efficient inference and learning. It adapts classification networks to fully convolutional networks and transfers learned representations to the segmentation task. It also defines a skip architecture that combines semantic information from a deep, coarse layer with appearance information from a shallow, fine layer to produce accurate and detailed segmentations. Many works later that make use of [101] are proposed later. Some works like [25, 35, 117, 125, 148] archive remarkable segmentation performance. [25] combines the responses at

FRAMEWORK	TRAINING	DESCRIPTION
Nest [51]	STDP/RSTDP	Nest focuses on the dynamics and structure of neural systems, and it is used in medical/biological applications but maps poorly to large datasets and deep learning.
Nengo [12] Nengo DL [134]	STDP ANN conversion backpropagation	NengoDL allows users to construct biologically detailed neural models, intermixed with deep learning elements bucked by TensorFlow and Keras [28].
SpykeTorch [118]	STDP/RSTDP	SpykeTorch is based on PyTorch [128] and simulates convolutional SNNs with at most one spike per neuron and the rank-order encoding scheme.
BindsNet [62]	STDP/RSTDP ANN conversion	BindsNet is also based on PyTorch targeting machine learning tasks. Currently, synapses are implemented without their own dynamics.
Slayer PyTorch [147]	backpropagation	Slayer PyTorch provides solutions for the temporal credit problem of spiking neurons that allows backpropagation of errors.

Table 2.6: Comparison of famous SNNs frameworks

the final Deep Convolutional Neural Networks (DCNNs) layer with a fully connected CRF to overcome the poor localization problem of deep networks. [35] proposes a method that achieves competitive accuracy but only requires easily obtained bounding box annotation. The basic idea is to iterate between automatically generating region proposals and training convolutional networks. These two steps gradually recover segmentation masks for improving the networks, and vise versa. [125] studies the more challenging problem of learning DCNNs for semantic image segmentation from either (1) weakly annotated training data such as bounding boxes or image-level labels or (2) a combination of few strongly labeled and many weakly labeled images, sourced from one or multiple dataset. [117] introduces a purely feed-forward architecture mapping small image elements (superpixels) to rich feature representations extracted from a sequence of nested regions of increasing extent. These regions are obtained by ”zooming out” from the superpixel all the way to scene-level resolution. This approach exploits statistical structure in the image and in the label space without setting up explicit structured prediction mechanisms, and thus avoids complex and expensive inference. In order to mitigates the limitations of the existing methods based on fully convolutional networks, [124] identifies detailed structures and handles objects in multiple scales natural by integrating deep deconvolution network and proposal-wise prediction. The deconvolution network is composed of deconvolution and unpooling layers, which identify pixelwise class labels and predict segmentation masks. To dealing with long range context, [148] introduce a global scene constraint to alleviate similar pixels they are indistinguishable from local context. They estimate the global potential in a non-parametric framework. For better global potential estimation, a large margin based CNN metric learning method is proposed. The final pixel class prediction is performed by integrating local and global beliefs. Recently, [99] explores ‘patch-patch’ context between image regions, and "patch-background" context. For learning from the patch-patch context, they formulate CRFs with CNN-based pairwise potential functions to capture semantic correlations between neighboring patches. Efficient piecewise training of the proposed deep structured model is then applied to avoid repeated expensive CRF inference for back propagation. For capturing the patch-background context, they use a network design with traditional multi-scale image input and sliding pyramid pooling.

Region-based Convolutional Neural Networks (R-CNN) is one of the important framework for the object detection and segmentation task [52] in this catorogy. R-CNN, the first generation of this family, applies the high-capacity deep CNN to classify given bottom-up region proposals. Due to the lack of labeled training data, it adopts a strategy of supervised pre-training for an auxiliary task followed by domain-specific fine-tuning. Then the ConvNet is used as a feature extractor

and the system is further trained for object detection with Support Vector Machines (SVM). Finally, it performs bounding-box regression. The method achieves high accuracy but is very time-consuming. The system takes a long time to generate region proposals, extract features from each image, and store these features in a hard disk, which also takes up a large amount of space. At testing time, the detection process takes 47s per image using VGG-16 network [150] implemented in GPU due to the slowness of feature extraction. In other words, R-CNN is slow because it processes each object proposal independently without sharing computation.

Fast R-CNN

Fast R-CNN [53] solves this problem by sharing the features between proposals. The network is designed to only compute a feature map once per image in a fully convolutional style, and to use ROI-pooling to dynamically sample features from the feature map for each object proposal. The network also adopts a multi-task loss, i.e. classification loss and bounding-box regression loss. Based on the two improvements, the framework is trained end-to-end. The processing time for each image significantly reduced to 0.3s. Fast R-CNN accelerates the detection network using the ROI-pooling layer. However the region proposal step is designed out of the network and, hence, still remains a bottleneck, which results in a sub-optimal solution and exhibits a dependence on the external region proposal methods.

Faster R-CNN

Faster R-CNN [136] addresses the problem with fast R-CNN by introducing the Region Proposal Network (RPN). An RPN is implemented in a fully convolutional style to predict the object bounding boxes and the objectness scores. In addition, the anchors, i.e. a set of predefined templates, are defined with different scales and ratios to achieve the translation invariance. The RPN shares the full-image convolution features with the detection network. Therefore the whole system is able to complete both proposal generation and detection computation within 0.2s using very deep VGG-16 model [150]. With a smaller Zeiler-Fergus (ZF) model [171], it can reach the level of real-time processing.

2.11.2 *Deep Reinforcement Learning-based Visual Navigation*

Visual navigation plays a key role when an artificial agent interacts with the environment to adapt to an environment and achieve advanced behaviors. Map-based methods, include simultaneous localization and mapping (SLAM) [157] and path planning [50] is a classical navigation method. SLAM system can be built on either laser sensor

or camera often paired with an IMU. Recent development in computer vision has focused on visual SLAM thanks to its capability of handle poor environment such as rain or snow. Visual navigation has been attracted by many researchers last couple years. Review on visual navigation can be found at [69]. In recent years, the success of Deep Reinforcement Learning has bring visual navigation into the next level.

3

METHODOLOGY

Our system contains two modules corresponding to scene understanding and robotic navigation. The first module is implemented as a Spiking Neural Network to carry out semantic segmentation to understand the scene in front of the robot. The second module provides a high-level navigation command to robot, which is considered as an agent and implemented by online reinforcement learning. The module was implemented with biologically plausible local learning rule that allows the agent to adopt quickly to the environment. We will first present the robot model that will be used in both simulation and real world.

3.1 ROBOT MODEL

The quadruped robot Lailaps was designed as the real world robotic platform to evaluate the proposed framework. Lailaps is a quadruped with 12 DoF with bilateral symmetry (Figure 3.1). The links of legs are connected to each other by rotary joints. Each leg is equipped with three servo motors that can be controlled through a Pulse Width Modulation (PWM) signal. The software PWM library pigpiod [176] was used for creating PWM signal from Raspberry Pi. The servos have a maximum torque 20.0 kg-cm, and may fail to follow the commands if the torque is insufficient.

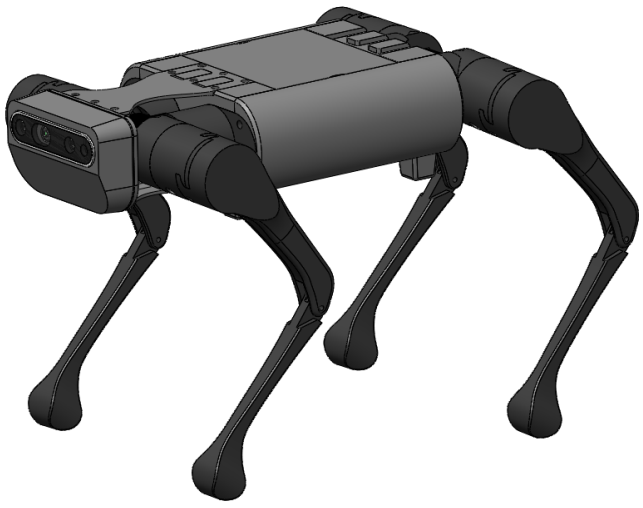


Figure 3.1: The overview of the physical model of the quadruped robot

The state of the robot is determined by its IMU and camera[1] inside the head.

3.1.1 *Leg Design*

The leg structure is presented in the Figure 3.2. The lower leg (calf and foot) is actuated by the servo motor inside the upper leg through the link rod to realize the slim appearance.

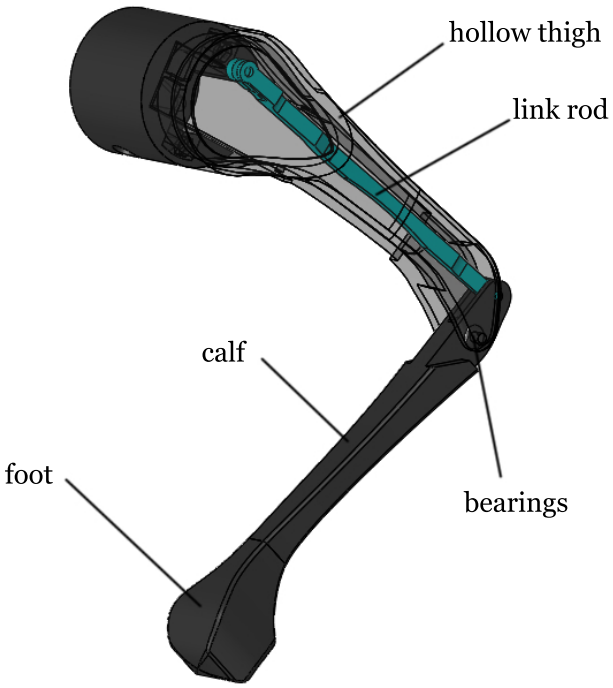


Figure 3.2: A structural design of a leg of the quadruped.

3.1.2 *Kinematic Analysis*

A set of basic homogeneous transformations for translation and rotation about the x, y, z axes is given by:

$$\boldsymbol{T}_{x,a} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

1 Intel RealSense D435i: https://www.intelrealsense.com/depth-camera-d435i/

$$\boldsymbol{T}_{y,b} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

$$\boldsymbol{T}_{z,c} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$\boldsymbol{R}_{x,\alpha} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

$$\boldsymbol{R}_{y,\beta} = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

$$\boldsymbol{R}_{z,\gamma} = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

Because of the bilateral symmetry of the robot, we will investigate the forward and inverse kinematics analysis of a single leg.

Forward Kinematics

We will use Denavit-Hartenberg convention for selecting frames of reference. The Denavit-Hartenberg parameters for forward kinematics of the leg are given in Table 3.1. In this convention, each homogeneous transformation A_i for associated i^{th} link and joint is represented as a product of four basic homogeneous transformations.

$$\boldsymbol{A_i} = \boldsymbol{R}_{z,\theta_i}\boldsymbol{T}_{z,d_i}\boldsymbol{T}_{x,a_i}\boldsymbol{R}_{x,\alpha_i} \tag{3.7}$$

where the four quantities $\theta_i, a_i, d_i, \alpha_i$ are representing the link length, link twist, link offset, and joint angle, respectively. The parameter a is

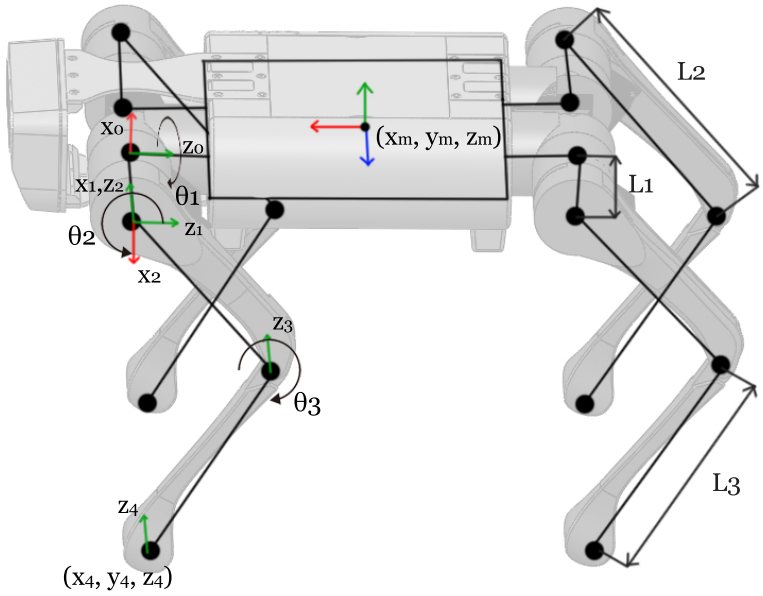


Figure 3.3: Quadruped state variables and forward kinematics

the distance between the common normal. Assuming a revolute joint, this is the radius about previous z axis. The angle α is the angle about common normal, from previous z axis to new z axis. The parameter d is the offset along previous z axis to the common normal. The angle θ is the angle about previous z axis, from previous x axis to the new x axis.

Link	a_i	α_i	d_{i+1}	θ_{i+1}
$0 \to 1$	L_1	0	0	$-\pi/2 + \theta_1^*$
$1 \to 2$	0	$-\pi/2$	0	$-\pi/2$
$2 \to 3$	L_2	0	0	$\pi/2 - \theta_2^*$
$3 \to 4$	L_3	0	0	θ_3^*

Table 3.1: Denavit-Hartenberg parameters of a leg. $*$ represents the variable.

The forward kinematic matrix T_4^0 is given in the following equation.

$$\boldsymbol{T_4^0} = \boldsymbol{A_1 A_2 A_3 A_4} \tag{3.8}$$

This results in a following matrix where the elements $m_{11} \sim m_{34}$ are listed in the Table 3.2.

$$\boldsymbol{T_4^0} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

m_{11}	-cos($\theta 1$) sin($\theta 2 - \theta 3$)
m_{12}	cos($\theta 1$) cos($\theta 2 - \theta 3$)
m_{13}	sin($\theta 1$)
m_{14}	L1 sin($\theta 1$) $-$ cos($\theta 1$)(L2 sin($\theta 2$) + L3 sin($\theta 2 - \theta 3$))
m_{21}	-sin($\theta 1$) sin($\theta 2 - \theta 3$)
m_{22}	sin($\theta 1$) cos($\theta 2 - \theta 3$)
m_{23}	-cos($\theta 1$)
m_{24}	-L1 cos($\theta 1$) $-$ sin($\theta 1$)(L2 sin($\theta 2$) + L3 sin($\theta 2 - \theta 3$))
m_{31}	-cos($\theta 2 - \theta 3$)
m_{32}	-sin($\theta 2 - \theta 3$)
m_{33}	0
m_{34}	-L2 cos($\theta 2$) $-$ L3 cos($\theta 2 - \theta 3$)

Table 3.2: The elements of the forward kinematic matrix

Inverse Kinematics

Following [178], the inverse kinematic analysis is performed using analytical methods to calculate angular position of each joints $(\theta_1, \theta_2, \theta_3)$. Given the desired foot position (x_4, y_4, z_4), required angle of each joint of a leg is calculated as follows:

$$\theta_1 = s \cdot atan2(-y_4, x_4) + s \cdot atan2(\sqrt{x_4^2 + y_4^2 - L_1^2}, -s \cdot L_1) \quad (3.10)$$

Considering the bilateral symmetry, the variable $s \in \{-1, 1\}$ accounts for a leg in left (-1) and right (1) side of the body.

$$\theta_2 = -atan2(z_4, \sqrt{x_4^2 + y_4^2 - L_1^2}) + atan2(L_3 \sin(\theta_3), L_2 + L_3 \cos(\theta_3)) \quad (3.11)$$

$$\theta_3 = d \cdot acos(\sqrt{x_4^2 + y_4^2 + z_4^2 - L_1^2 - L_2^2 - L_3^2}/(2L_2L_3)) \quad (3.12)$$

Here the variable $d \in \{-1, 1\}$ accounts for plantigrade (-1) and digitigrade (1) mammal-like leg orientation of the robot. Since Lailaps has leg orientation of digitigrade mammal type, $d = 1$ was used.

Robot Kinematics

We define the rotational matrix and the transformation matrix given the rotation (ϕ, ψ, ω) and translation (x_m, y_m, z_m) of the body coordinate system of the robot as follows:

$$\boldsymbol{R_m} = \boldsymbol{R_{x,\phi}} \boldsymbol{R_{y,\psi}} \boldsymbol{R_{z,\omega}} \quad (3.13)$$

$$\boldsymbol{T_m} = \boldsymbol{R_m} \times \begin{bmatrix} 1 & 0 & 0 & x_m \\ 0 & 1 & 0 & y_m \\ 0 & 0 & 1 & z_m \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.14}$$

Then the base orientation and translation and base of the each leg can be related as:

$$\boldsymbol{T_{FR}} = \boldsymbol{T_m} \times \begin{bmatrix} \cos(\pi/2) & 0 & \sin(\pi/2) & L/2 \\ 0 & 1 & 0 & 0 \\ -\sin(\pi/2) & 0 & \cos(\pi/2) & -W/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.15}$$

$$\boldsymbol{T_{FL}} = \boldsymbol{T_m} \times \begin{bmatrix} \cos(\pi/2) & 0 & \sin(\pi/2) & L/2 \\ 0 & 1 & 0 & 0 \\ -\sin(\pi/2) & 0 & \cos(\pi/2) & W/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.16}$$

$$\boldsymbol{T_{BR}} = \boldsymbol{T_m} \times \begin{bmatrix} \cos(-\pi/2) & 0 & \sin(-\pi/2) & -L/2 \\ 0 & 1 & 0 & 0 \\ -\sin(-\pi/2) & 0 & \cos(-\pi/2) & -W/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

$$\boldsymbol{T_{BL}} = \boldsymbol{T_m} \times \begin{bmatrix} \cos(-\pi/2) & 0 & \sin(-\pi/2) & -L/2 \\ 0 & 1 & 0 & 0 \\ -\sin(-\pi/2) & 0 & \cos(-\pi/2) & W/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.18}$$

where L and W are the separation of hip joint along x and y axis respectively.

3.1.3 *Simulation Model*

We developed simulation model of the robot using PyBullet [33] physics engine for safe and rapid prototyping of motion generation and testing. PyBullet is a Python module that wraps the new Bullet C-API for robotics simulation and machine learning, with a focus on sim-to-real transfer. The robot model was expressed in URDF format [133] and imported to the environment. The simulation model is shown in Figure 3.4.

Section A.1 for more details.

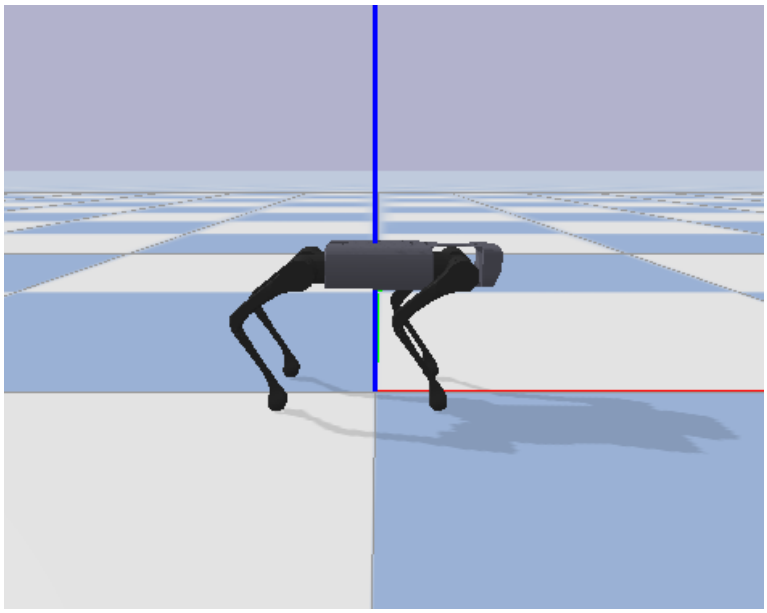


Figure 3.4: A quadruped model in the simulator

3.1.4 *Foot Trajectory Generation*

The leg movement of a quadruped robot can be divided into two phases: the stance phase and the swing phase. We separately designed the foot trajectory based on the phase.

Stance Phase Trajectory

Cosine function was used in the trajectory of the stance phase since it provides a smooth trajectory. This method have been used in foot trajectory generation in quadruped robots [91, 109] . The z-directional formula of stance phase is shown below:

$$T_z^{ST}(t) = -0.001\cos(\frac{\pi}{2}(1-t)) \tag{3.19}$$

where $t \in (0, 1]$ represent the time-step in stance phase.

Swing Phase Trajectory

The trajectory of the swing phase can be obtained by Bézier curve [172], which is a smooth parametric curve controlled by several points expressed as follows:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i P_i \tag{3.20}$$

where P_i is the i^{th} fitting point and t . This curve has been widely used in swing phase trajectory generation of quadruped robots [70, 91]. We

obtained a smooth foot trajectory by ten control points as shown in Table 3.3.

P_n	$x[m]$	$z[m]$
P_0	-0.05	0
P_1	-0.06	0
P_2	-0.07	0.05
P_3	-0.07	0.05
P_4	0	0.05
P_5	0	0.05
P_6	0.07	0.06
P_7	0.07	0.06
P_8	0.06	0
P_9	0.05	0

Table 3.3: The 10 control points of the Bézier curve. Note that the z values are scaled by a desired foot height.

3.2 SPIKING SCENE UNDERSTANDING

If a robot is to interact with its environment, the robot must be able to sense its surrounding environment. This could be done through various types of sensors or collections of sensors. In this thesis, we will present a SNN based image segmentation model for scene understanding. Image segmentation is the process of partitioning an image into multiple homogeneous regions that share some common properties, e.g. intensity, color or texture. Image segmentation plays an important role in various areas of image processing, such as object detection and classification, action classification, scene understanding, and other visual information analysis processes. Convolutional neural networks (CNNs), in this context, are one of the most powerful sensing modalities that currently exists. We employed fully-convolutional deep neural network to extract such pixel information and later convert to SNN by replacing the activation functions to the spiking neurons.

3.2.1 *Network Structure*

The overview of the ANN segmentation model is provided in Figure 3.5. The detailed parameters of the network can be found on Figure A.1. The network has total of 19 convolutional layers followed by ReLU activation function for the later conversion. Although Kim et al. presented the way to convert more powerful activation function LeakyReLU [167] into spiking layer by introducing "negative spikes"

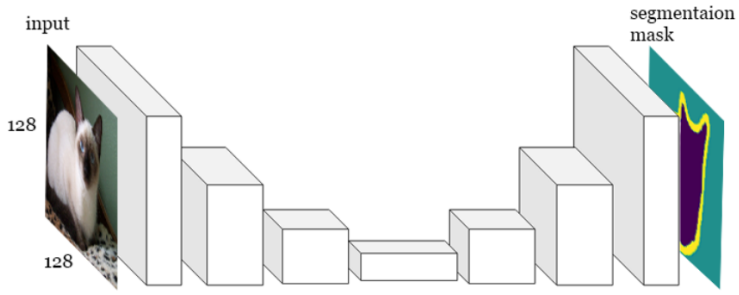


Figure 3.5: The network structure of the segmentation model

[79], we did not employ this activation function for simplicity and biological plausibility.

3.2.2 *Weight Scaling*

In order to preserve the information encoded within a layer, the parameters of a layer need to be scaled jointly [170]. This could be done by scaling the weights based on the data-based normalization scheme [41]. This technique relies on the linearity of the ReLU activation function in ANN.

$$W^l_{SNN} \leftarrow W^l_{ANN} \frac{\lambda^{l-1}}{\lambda^l} \tag{3.21}$$

where W^l is the weight of layer l and λ^l is the maximum ReLU activation of layer l.

3.2.3 *Information Coding*

Although the brain is known to encode external information into binary events, the way information is encoded with such spikes being one of the big unresolved challenges. According to Zambrano and Bohte, we can use analog input values in the very first hidden layer, and compute with spikes from there on [170]. This method works effectively in the low-activation range in ANN units, where under-sampling in spiking neurons usually occurs [139]. Following these observations, we did not converted analog input activations or RGB values into Poisson firing rates, instead we fed the normalized RGB values into the first convolutional layer directly for a given time steps inorder to obtain the spike activations for consecutive layers.

3.3 SPIKE BASED ONLINE LEARNING

3.3.1 *Simulation Environment*

In order to apply the reinforcement learning methods, a standalone module that adopts the OpenAI Gym [22] interface was built on top of the PyBullet environment. Within the Gym environment, the agent perceives the world through a robot's sensors (simulated LiDAR, camera, IMU, etc.) and interacts with the world through its actuators.

In the navigation environment, an agent needs to move towards a series of goal positions. When the agent reaches the goal, the goal location is randomly reset to elsewhere, while keeping the rest of the obstacles' layout the same. The sparse reward component is attained on achieving a goal position. The dense reward component gives a bonus for moving towards the goal. Whenever the robot collides with the obstacles, negative sparse reward, or punishment, is given to the agent. In this environment, an agent can give a high-level commands (move forward/backward, rotate clockwise/counter-clockwise) that is executed by the robot platform. The robot is equipped with LiDAR and/or RGBD camera (Figure 3.6).

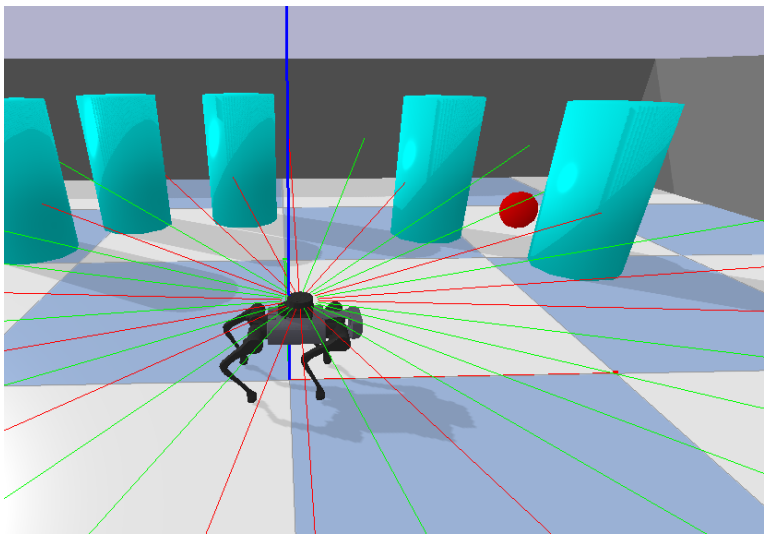


Figure 3.6: The snapshot of the navigation gym environment. The blue cylinders are the obstacles and the red sphere is the goal to reach.

3.3.2 *Control Architecture*

The overall control architecture is represented in Figure 3.7. The observations are encoded into the sensory neurons, which are the ensemble of LIF neurons, and the synaptic weight between the sensory neurons and the basal ganglia was adjusted in reinforcement learning paradigm through online manner. The model learns the synaptic

weight through PSE learning rule based on the observed reward from the environment.

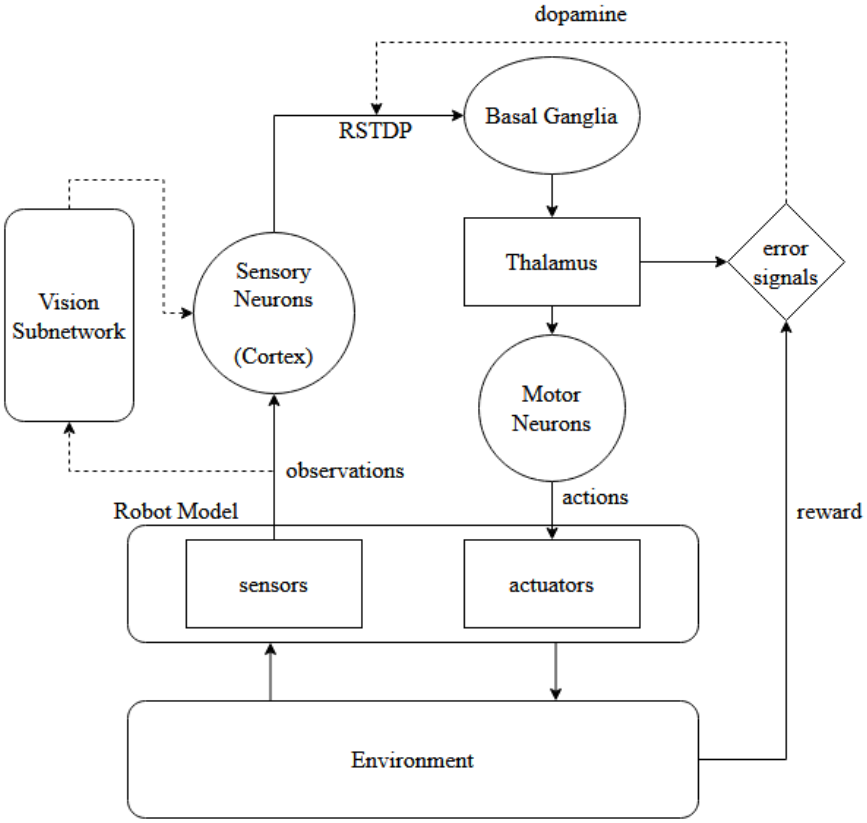


Figure 3.7: The architecture of high-level navigation controller that follows the Agent-Environment interaction in reinforcement learning.

3.3.3 *Vision Subnetwork*

Vision subnetwork is intended to encode visual information from the robot camera. The segmentation model presented in the previous section could be employed here but we used more shallower network for real-time computation. U-Net architecture [137], which is noted in (medical) image segmentation applications, features in skip connections between the mirrored layers in the encoder and decoder, allowing successive convolution layers to learn a more precise output based on the information of high resolution features as shown in Figure 3.8. Unlike the original purpose of using U-Net to conduct semantic segmentation, we exploit its architecture to obtain an encoded visual information. The subnetwork was implemented in ANN to be trained to reproduce the input image, allowing the bottleneck features to be the encoded representation of the input image.

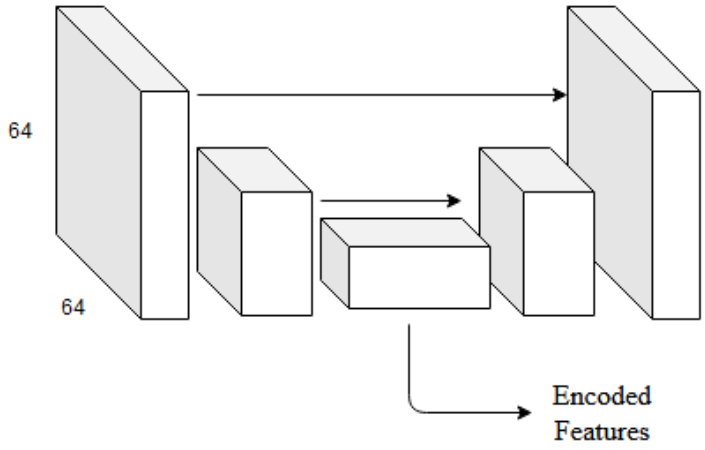


Figure 3.8: The network structure of the segmentation model

Basal Ganglia

Basal Ganglia is responsible for action selection [4]. Action selection consists of choosing one action to perform out of the many actions in an organism's repertoire. In our case it will be the four actions that the agent can take, i.e., move forward/backward, and rotate clockwise/counter-clockwise. According to Nambu, Tokuno, and Takada, the basic components of basal ganglia are the striatum (striatal D1 and D2), the subthalamic nucleous (STN), the globus pallidus internal (GPi) / substantia nigra pars reticulata (SNr), and the globus pallidus external (GPe) [120]. Gurney, Prescott, and Redgrave proposed non-spiking basal ganglia model [57] with one neuron per area of the basal ganglia to represent each action. The model was later extended to spiking version by Stewart, Choo, and Eliasmith by adapting the rate model into spiking neurons (LIF model) [154]. The basal ganglia model is primarily defined by its winner-take-all function and outputs approximately zero at the dimension with the largest value and negative elsewhere.

Thalamus

The thalamus inhibits non-selected actions and intended to work in tandem with a basal ganglia network. The thalamus helps the discretization of the output of the basal ganglia network. In order to suppress low responses and strengthen high responses, a constant bias is added to each dimension, and dimensions mutually inhibit each other. Additionally, the ensemble representing each dimension is created with positive encoders and can be assigned positive x-intercepts to threshold low responses.

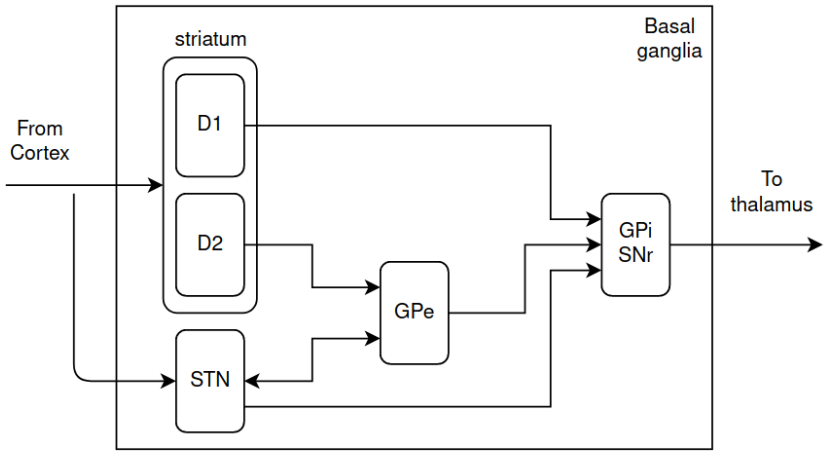


Figure 3.9: The architecture of basal ganglia for action selection (based on [57, 154]).

4

EXPERIMENTS AND RESULTS

4.1 SPIKING SCENE UNDERSTANDING

Since the direct training of the deep SNN is still challenging, we followed the ANN-to-SNN conversion strategy for this work as descried in the previous chapter. As a first step, the fully convolutional deep neural network with regular ReLU activation functions was trained using standard DNN framework. This allows us to validate the network architecture to perform the segmentation task sufficiently.

4.1.1 *Dataset and Metrics*

Dataset: The Oxford-IIIT Pet dataset [126] was used for this experiment. The dataset consists 37 category namely, 25 categories of dogs and 12 categories of cats. Each class with roughly consists of 200 images. The images have a large variations in scale, pose and lighting. All images have an associated ground truth annotation of breed, head ROI, and pixel level trimap (i.e. background, edge and object region) segmentation. The dataset is given in Fig.4.1 and its annotations is shown in Fig.4.2. All the data was resized to $(128, 128)$ and normalized to 0 to 1.

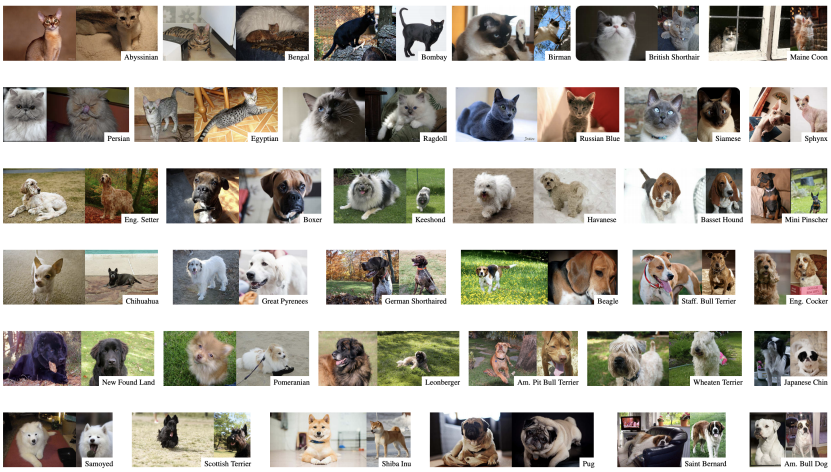


Figure 4.1: Oxford-IIIT Pet dataset of 37 category of dogs and cats.

Metrics: To train a DNN, the loss function, which is known as cost function, plays a significant role. Loss function is to measure the average (expected) divergence between the output of the network (P) and the actual function (T) being approximated, over the entire

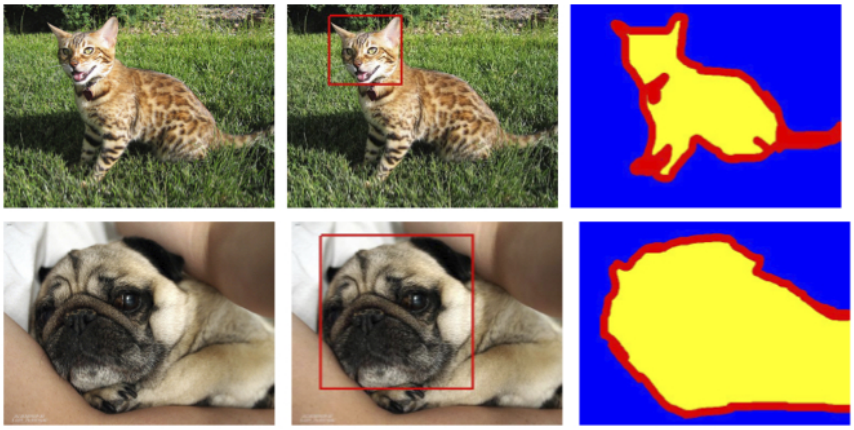


Figure 4.2: Some annotations examples from Oxford-IIIT Pet Dataset. Each image has its own class label (first column), tight bounding box (ROI) around the head of the animal (second column) and a pixel level foreground-boundary-background segmentation.

domain of the input, sized $m \times n$. We denote i as index of each pixel in an image spatial space $N = m \times n$. The label of each class is written as c in C classes. Herein, we briefly review the some common loss functions.

In this thesis, we use Cross Entropy (CE) Loss which is a widely used pixel-wise distance to evaluate the performance of classification or segmentation model. In CE loss function, the output from softmax layer (P) is classified and evaluated against the groundtruth (T). For binary segmentation, CE loss is expressed as Binary-CE (CE) loss function as follows:

$$\mathcal{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N}[T_i \ln(P_i) + (1 - T_i)\ln(1 - P_i)] \tag{4.1}$$

The standard CE loss has well-known drawbacks in the context of highly unbalanced problems. It achieves good performance on a large training set with balanced classes. For unbalanced data, it however typically results in unstable training and leads to decision boundaries biased towards the majority classes. To deal with the imbalanced-data problem, two variants of the standard CE loss, Weighted CE (WCE) loss and Balanced CE (BCE) loss are proposed to assign weights to the different classes.

To deal with the imbalanced-data problem, two variants of the standard CE loss, Weighted CE (WCE) loss and Balanced CE (BCE) loss are proposed to assign weights to the different classes. WCE, BCE losses assign more importance to the rare labels and defined as

$$WCE(T, P) = -\frac{1}{N}\sum_{i=1}^{N}[\beta T_i \ln(P_i) + \gamma(1 - T_i)\ln(1 - P_i)] \tag{4.2}$$

where $\beta > 1$ is to decrease the number of false negatives and where $\beta < 1$ is to decrease the number of false positives. In WCE loss, $\gamma = 1$ whereas $\gamma = 1 - \beta$ in BCE.

4.1.2 *Training of ANN*

The non-spiking convolutional neural network was trained with Adam optimizer [80] through backpropagation. The hyper-parameters was set as default of Keras framework. The model was trained on a machine with Intel Core i7-7700K CPU @ 4.20GHz × 8 with 32 GB RAM and GeForce GTX 1070/PCIe/SSE2 running Ubuntu 20.04. Regarding the conversion, we target the spiking neurons to have a firing rates of $150 \sim 200Hz$ to ensure the spiking because too low firing rate would result in a generation of no activity especially for the deep network architecture.

4.1.3 *Experimental Result*

The Figure 4.3 shows some segmentation result done by the converted model. Although the details are not segmented well, the converted SNN model can still capture the overall structure of an object within the image. We use intersection over union metric (IoU) to evaluate the accuracy of the predicted masks. IoU is is evaluated as:

$$IoU(b_A, b_B) = \frac{b_A \cap b_B}{b_A \cup b_B} \tag{4.3}$$

where the b_A and b_A represents the ground truth and predicted mask respectively.
The ANN model presents the IoU of 0.803 on the dataset while the converted SNN model presents the IoU of 0.785. The degrade in accuracy is thought to be mainly because of the conversion. As for the segmentation model, the accuracy could be improved via: (1) training better ANN the before conversion, (2) improving the conversion strategy. Because of the complex routing in state of the art ANN models, there are still some obstacles for the ANN to SNN conversion. Adding a firing rate regularization term to the cost function that penalizes neurons firing outside of the desired range is one option. Networks could also be trained offline as described and then fine-tuned online using an STDP rule to help further reduce errors associatedwith converting from rate-based to spike-based networks, while avoiding difficulties with training a network in spiking neurons from scratch.

4.2 ROBOT NAVIGATION

In order to provide the robot a high-level command, we implemented the agent capable of online reinforcement learning. The network architecture was discussed in the previous chapter and the overall architecture is shown in Figure 3.7.

4.2.1 *Performance*

Figure 4.4 shows the (x, y) trajectory of the robot moving throughout the environment to five consecutive targets in the environment. As shown in Figure 4.4, without the learning capabilities, the robot failed to reach the objective. When the online learning is activated, robot wonders around and searched for the sensory signals that gives more reward from the environment. The robot gets better in reaching the objective as time passes. This process only takes about 2 minuets, showing the high-adaptability to the sensory data streams. In the real-world robotic applications, the reward could be given by a human teaching what is expected.

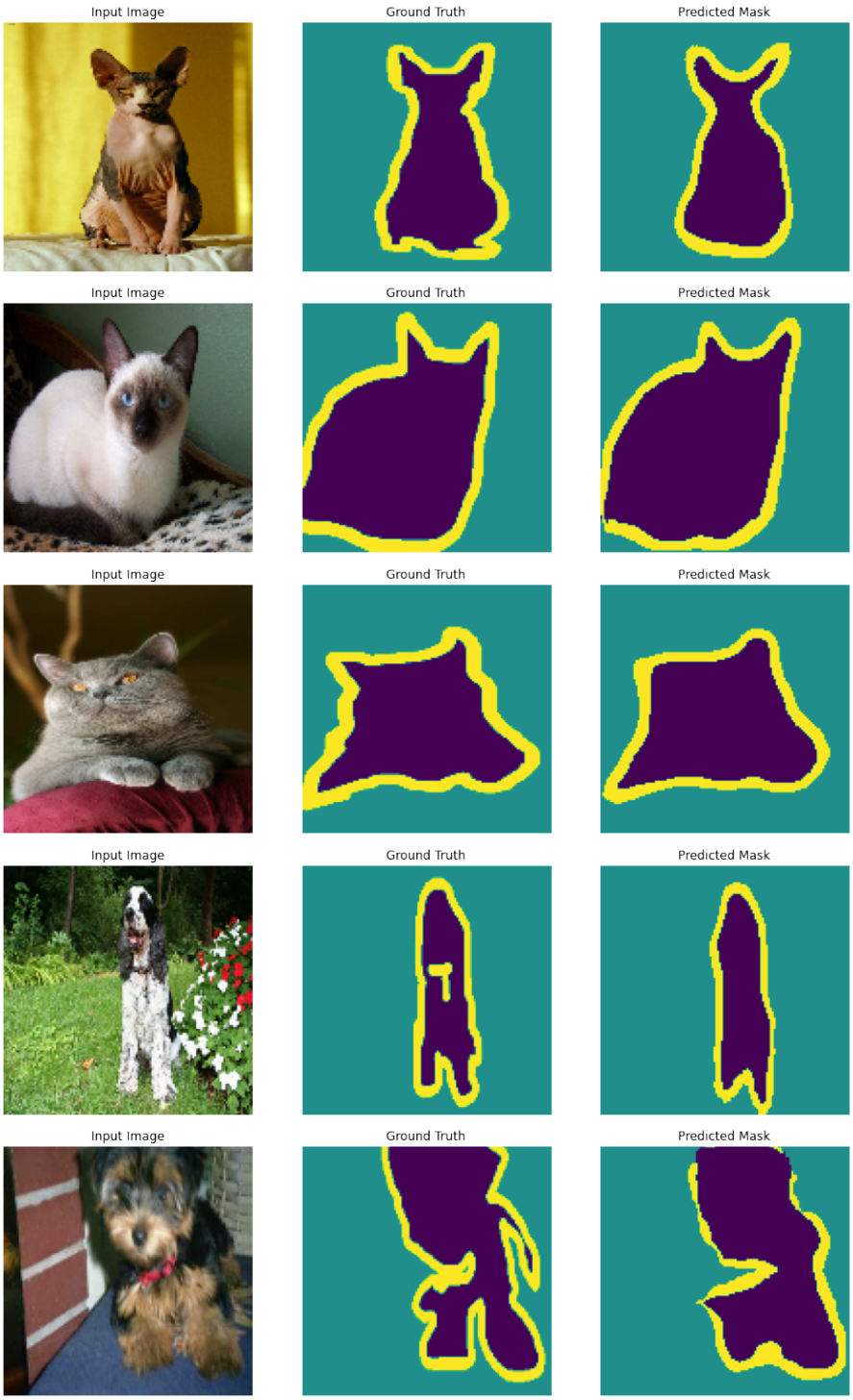


Figure 4.3: Segmentation results obtained by the converted SNN.

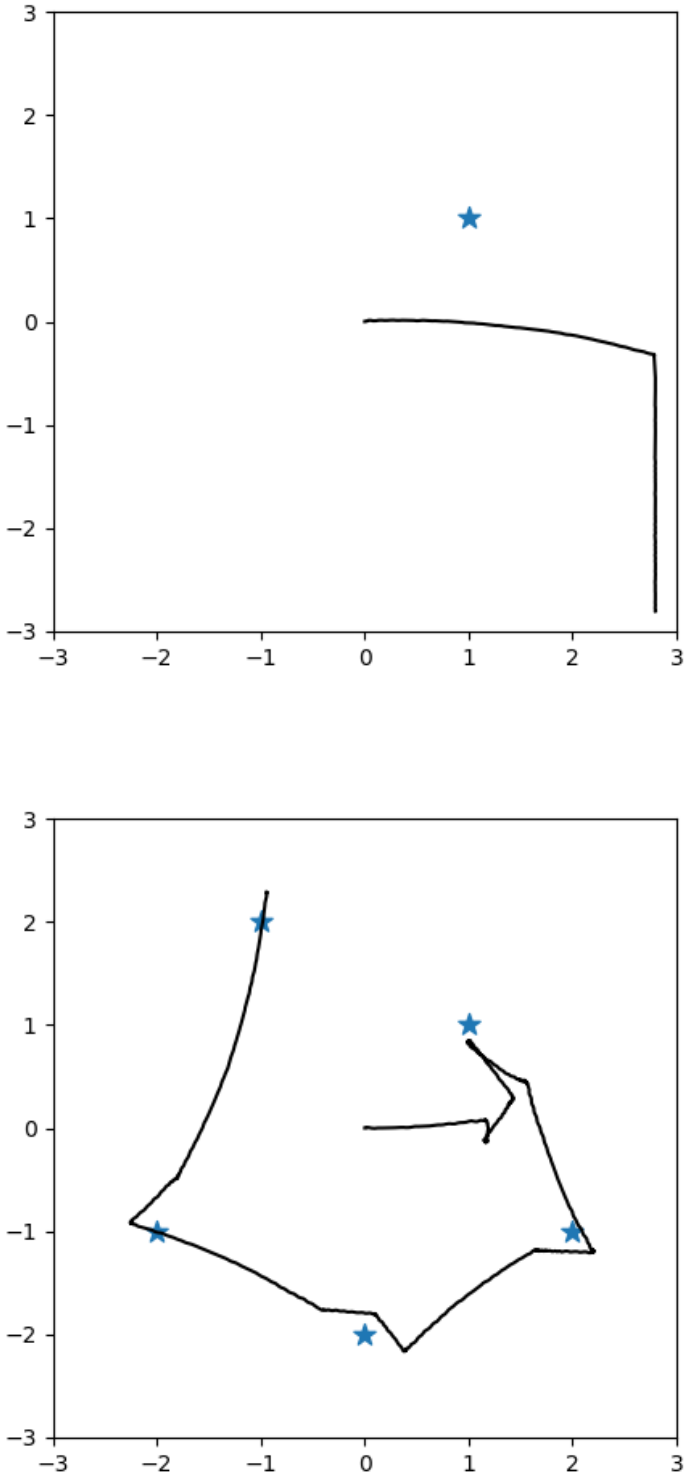


Figure 4.4: TOP: Path of the robot without online learning. The robot moved forward only to hit the wall and stuck at the corner. BOTTOM: Path of the robot with online learning. The robot gradually gets better in pursuing the objective.

5 CONCLUSIONS

This project aimed to lay the foundation for presenting SNN-based control strategies for robots. We explored the use of SNNs in robotics in favour of brains' exhibition of a remarkable cognitive performance in real-world tasks. Our experimental result assures the initial step of the development for a fully autonomous robot. In summary, the main contributions of this work are as follows:

A. Implemented a spiking image segmentation model with ANN to SNN conversion, which showed a potential use of SNN in computer vision for robotic perception.

B. Implemented a online reinforcement learning in spiking neurons with biologically plausible synaptic plasticity that allows to quickly adopt to the environment.

There is clearly significant room for improvement and extension to this work. It is the hopes of this work that presented result promotes the research in both SNN and robotics in order to realize the animal-like intelligence in the near future.

5.1 FUTURE WORK

To further develop this work and enable fully autonomous control of a robot with spiking neural network, we have to overcome the hardware constraints.

5.1.1 *Low-level Control*

This work focused in high-level control of a quadruped robot but we hope to extend this SNN framework to the low-level control of the robot as well. This includes the gait pattern generation based on the central pattern generators (CPGs) and reflex systems.

5.1.2 *Neuromorphic Hardware*

While every simulation of SNN in this work was done on a classical hardware, we want to deploy the network on the event-driven hardware in order to embrace full advantage of SNN. This means that further work can be done both on the software and the hardware side to create energy efficient and adoptive model of SNN for robotic applications. This work could be extended with the development on

neuromorphic chip such as Intel Loihi research chip, which contains 130,000 neurons optimized for spiking neural networks, along with an dynamic vision sensors (DVS), also known as an event camera.

5.1.3 *ROS Development*

Integration of this work with Robot Operating System (ROS) is the another future work. In ROS eco-system, many of the basic framework for robotic software development has already been completed. It would be beneficial to provide SNN package for robotic navigation and control for the future development in neurorobotics. Once the ROS development is completed, we hope to employ the SNNs in various kinds of real robots.

A

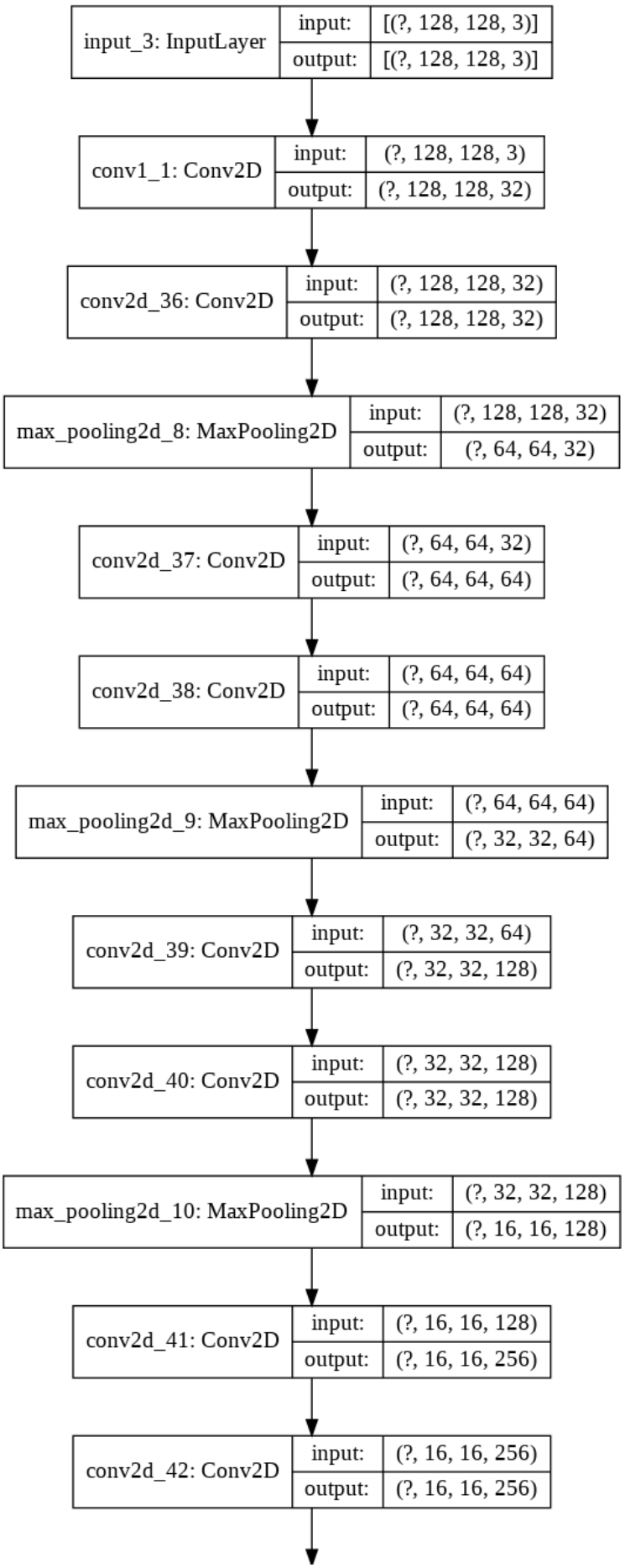
APPENDIX

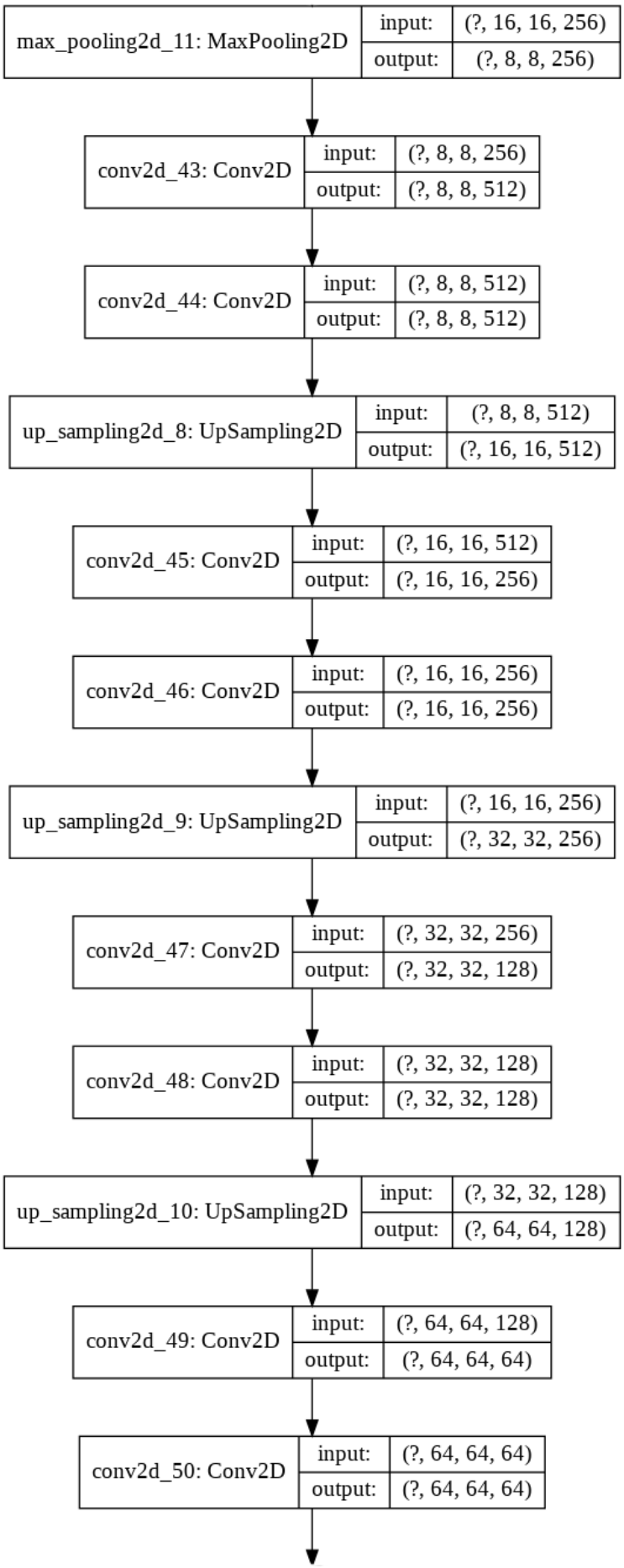
A.1 CREATING URDF FROM SOLIDWORKS MODEL

The Unified Robotic Description Format (URDF) is an XML file format used in ROS to describe all elements of a robot in tree structure. In order to create the URDF file from a SolidWorks CAD model, the SolidWorks to URDF exporter (`http://wiki.ros.org/sw_urdf_exporter`), a SolidWorks add-in that allows SolidWorks Parts and Assemblies to be converted into a URDF file, can be utilized.

A.2 STRUCTURE OF SPIKING NEURAL NETWORKS

A.3 GALLERY


input_3: InputLayer
input: [(?, 128, 128, 3)]
output: [(?, 128, 128, 3)]
conv1_1: Conv2D
input: (?, 128, 128, 3)
output: (?, 128, 128, 32)
conv2d_36: Conv2D
input: (?, 128, 128, 32)
output: (?, 128, 128, 32)
max_pooling2d_8: MaxPooling2D
input: (?, 128, 128, 32)
output: (?, 64, 64, 32)
conv2d_37: Conv2D
input: (?, 64, 64, 32)
output: (?, 64, 64, 64)
conv2d_38: Conv2D
input: (?, 64, 64, 64)
output: (?, 64, 64, 64)
max_pooling2d_9: MaxPooling2D
input: (?, 64, 64, 64)
output: (?, 32, 32, 64)
conv2d_39: Conv2D
input: (?, 32, 32, 64)
output: (?, 32, 32, 128)
conv2d_40: Conv2D
input: (?, 32, 32, 128)
output: (?, 32, 32, 128)
max_pooling2d_10: MaxPooling2D
input: (?, 32, 32, 128)
output: (?, 16, 16, 128)
conv2d_41: Conv2D
input: (?, 16, 16, 128)
output: (?, 16, 16, 256)
conv2d_42: Conv2D
input: (?, 16, 16, 256)
output: (?, 16, 16, 256)


max_pooling2d_11: MaxPooling2D
input: (?, 16, 16, 256)
output: (?, 8, 8, 256)
conv2d_43: Conv2D
input: (?, 8, 8, 256)
output: (?, 8, 8, 512)
conv2d_44: Conv2D
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
up_sampling2d_8: UpSampling2D
input: (?, 8, 8, 512)
output: (?, 16, 16, 512)
conv2d_45: Conv2D
input: (?, 16, 16, 512)
output: (?, 16, 16, 256)
conv2d_46: Conv2D
input: (?, 16, 16, 256)
output: (?, 16, 16, 256)
up_sampling2d_9: UpSampling2D
input: (?, 16, 16, 256)
output: (?, 32, 32, 256)
conv2d_47: Conv2D
input: (?, 32, 32, 256)
output: (?, 32, 32, 128)
conv2d_48: Conv2D
input: (?, 32, 32, 128)
output: (?, 32, 32, 128)
up_sampling2d_10: UpSampling2D
input: (?, 32, 32, 128)
output: (?, 64, 64, 128)
conv2d_49: Conv2D
input: (?, 64, 64, 128)
output: (?, 64, 64, 64)
conv2d_50: Conv2D
input: (?, 64, 64, 64)
output: (?, 64, 64, 64)

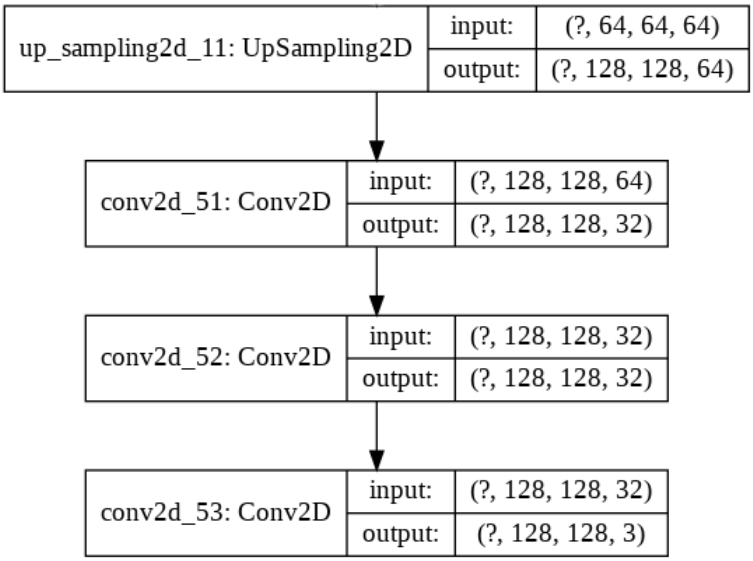


Figure A.1: Network Structure of ANN used for segmentation


MECHANICAL ENGINEERING
1998

BIBLIOGRAPHY

[1] Pieter Abbeel, Adam Coates, and Andrew Y. Ng. "Autonomous Helicopter Aerobatics through Apprenticeship Learning." In: *The International Journal of Robotics Research* 29.13 (2010), pp. 1608–1639.

[2] L. Abbott. "Lapicque's introduction of the integrate-and-fire model neuron (1907)." In: *Brain Research Bulletin* 50.5 (1999), pp. 303 –304. ISSN: 0361-9230. DOI: https://doi.org/10.1016/S0361-9230(99)00161-6. URL: http://www.sciencedirect.com/science/article/pii/S0361923099001616.

[3] Stephan Alaniz. "Deep Reinforcement Learning with Model Learning and Monte Carlo Tree Search in Minecraft." In: *Conference on Reinforcement Learning and Decision Making*. 2018.

[4] J. R. Anderson, Michael D. Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. "An Integrated Theory of the Mind." In: *Psychological Review* 111.4 (2004), pp. 1036–1050.

[5] O. Andersson, F. Heintz, and P. Doherty. "Model-Based Reinforcement Learning in Continuous Environments Using Real-Time Constrained Optimization." In: *AAAI*. 2015.

[6] Tatsuya Aoki, Tomoaki Nakamura, and Takayuki Nagai. "Learning of Motor Control from Motor Babbling**This research is supported by CREST, JST." In: *IFAC-PapersOnLine* 49.19 (2016). 13th IFAC Symposium on Analysis, Design, and Evaluation ofHuman-Machine Systems HMS 2016, pp. 154 –158. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2016.10.478. URL: http://www.sciencedirect.com/science/article/pii/S2405896316320687.

[7] F. Azevedo, L. Carvalho, L. Grinberg, J. Farfel, R. Ferretti, R. Leite, W. Filho, R. Lent, and S. Herculano-Houzel. "Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain." In: *The Journal of comparative neurology* 513 5 (2009), pp. 532–41.

[8] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. "GA3C: GPU-based A3C for Deep Reinforcement Learning." In: *CoRR* abs/1611.06256 (2016).

[9] J. A. Bagnell and J. G. Schneider. "Autonomous helicopter control using reinforcement learning policy search methods." In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 2. 2001, pp. 1615–1620.

[10] J. Bagnell. "Learning Decision: Robustness, Uncertainty, and Approximation." In: (Apr. 2012).

[11] Trevor Bekolay, Carter Kolbeck, and Chris Eliasmith. "Simultaneous unsupervised and supervised learning of cognitive functions in biologically plausible spiking neural networks." In: Jan. 2013, pp. 169–174.

[12] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence Stewart, Daniel Rasmussen, Xuan Choo, Aaron Voelker, and Chris Eliasmith. "Nengo: a Python tool for building large-scale functional brain models." In: *Frontiers in Neuroinformatics* 7.48 (2014), pp. 1–13. ISSN: 1662-5196. DOI: 10.3389/fninf.2013.00048.

[13] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning Long-Term Dependencies with Gradient Descent is Difficult." In: *IEEE Trans. Neural Networks* 5.2 (1994), pp. 157–166.

[14] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen. "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations." In: *Proceedings of the IEEE* 102.5 (2014), pp. 699–716.

[15] M. Bernardo, J. Budd, R. Champneys, and P. Kowalczyk. *Piecewise-smooth dynamical systems: theory and applications*. Applied Mathematical Sciences. Springer, 2008. ISBN: 9781846280399.

[16] Shalabh Bhatnagar. "An actor-critic algorithm with function approximation for discounted cost constrained Markov decision processes." In: *Systems Control Letters* 59.12 (2010), pp. 760 –766. ISSN: 0167-6911. DOI: https://doi.org/10.1016/j.sysconle.2010.08.013. URL: http://www.sciencedirect.com/science/article/pii/S0167691110001246.

[17] Shalabh Bhatnagar, Richard S. Sutton, Mohammad Ghavamzadeh, and Mark Lee. "Natural actor-critic algorithms." In: *Automatica* 45.11 (2009), pp. 2471 –2482.

[18] G. Bi and M. Poo. "Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type." In: *Journal of Neuroscience* 18.24 (1998), pp. 10464–10472. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.18-24-10464.1998. URL: https://www.jneurosci.org/content/18/24/10464.

[19] Zhenshan Bing, Claus Meschede, Florian Röhrbein, Kai Huang, and Alois C. Knoll. "A Survey of Robotics Control Based on Learning-Inspired Spiking Neural Networks." In: *Frontiers in Neurorobotics* 12 (2018), p. 35.

[20] "Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks." In: *Pattern Recognition* 94 (2019), pp. 87 –95.

[21] J. Boedecker, J. T. Springenberg, J. Wülfing, and M. Riedmiller. "Approximate real-time optimal control based on sparse Gaussian process models." In: *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. 2014, pp. 1–8.

[22] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. "OpenAI Gym." In: *arXiv e-prints* (2016). arXiv: 1606.01540 [cs.LG].

[23] Robin Brügger, Christian F. Baumgartner, and Ender Konukoglu. "A Partially Reversible U-Net for Memory-Efficient Volumetric Image Segmentation." In: *arXiv e-prints*, arXiv:1906.06148 (June 2019), arXiv:1906.06148. arXiv: 1906.06148 [cs.CV].

[24] S. Cavallari, S. Panzeri, and A. Mazzoni. "Comparison of the dynamics of neural interactions between current-based and conductance-based integrate-and-fire recurrent networks." In: *Frontiers in Neural Circuits* 8.MAR (2014), pp. 1–23. ISSN: 16625110. DOI: 10.3389/fncir.2014.00012.

[25] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. "Semantic image segmentation with deep convolutional nets and fully connected CRFs." In: *ICLR*. 2015, abc.

[26] Ling-Hui Chen, Tuomo Raitio, Cassia Valentini-Botinhao, Junichi Yamagishi, and Zhen-Hua Ling. "DNN-based stochastic postfilter for HMM-based speech synthesis." In: *INTERSPEECH*. 2014, pp. 1954–1958.

[27] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation." In: *CoRR* abs/1406.1078 (2014).

[28] François Chollet et al. *Keras*. https://github.com/fchollet/keras. 2015.

[29] Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, KyungHyun Cho, and Yoshua Bengio. "Attention-Based Models for Speech Recognition." In: *CoRR* abs/1506.07503 (2015).

[30] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. "Model-Based Reinforcement Learning via Meta-Policy Optimization." In: *CoRR* abs/1809.05214 (2018).

[31] Adam Coates, Pieter Abbeel, and Andrew Ng. "Apprenticeship Learning for Helicopter Control." In: *Commun. ACM* 52 (July 2009), pp. 97–105. DOI: 10.1145/1538788.1538812.

[32] Rémi Coulom. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search." In: *Computers and Games*. Ed. by H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 72–83. ISBN: 978-3-540-75538-8.

[33] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. `http://pybullet.org`. 2016–2019.

[34] G. Cybenko. "Approximation by superpositions of a sigmoidal function." In: *Mathematics of Control, Signals, and Systems (MCSS)* 2.4 (1989), pp. 303–314. ISSN: 0932-4194. DOI: `10.1007/BF02551274`.

[35] J. Dai, K. He, and J. Sun. "Boxsup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation." In: *ICCV*. 2015.

[36] M. Davies et al. "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning." In: *IEEE Micro* 38.1 (2018), pp. 82–99.

[37] Kristopher De Asis, J Fernando Hernandez-Garcia, G Zacharias Holland, and Richard S Sutton. "Multi-step reinforcement learning: A unifying algorithm." In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[38] M. P. Deisenroth, P. Englert, J. Peters, and D. Fox. "Multi-task policy search for robotics." In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 3876–3881.

[39] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[40] B. Depraetere, M. Liu, G. Pinte, I. Grondman, and R. BabuÅ¡ka. "Comparison of model-free and model-based methods for time optimal hit control of a badminton robot." In: *Mechatronics* 24.8 (2014), pp. 1021 –1030.

[41] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. Liu, and M. Pfeiffer. "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing." In: *2015 International Joint Conference on Neural Networks (IJCNN)*. 2015, pp. 1–8. DOI: `10.1109/IJCNN.2015.7280696`.

[42] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. "Long-term Recurrent Convolutional Networks for Visual Recognition and Description." In: *CoRR* abs/1411.4389 (2014).

[43] Michael Egmont-Petersen, Dick de Ridder, and Heinz Handels. "Image processing with neural networks a review." In: *Pattern recognition* 35.10 (2002), pp. 2279–2301.

[44] A. El-Fakdi and M. Carreras. "Policy gradient based Reinforcement Learning for real autonomous underwater cable tracking." In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2008, pp. 3635–3640.

[45] Christian Etmann, Rihuan Ke, and Carola-Bibiane Schönlieb. "iUNets: Fully invertible U-Nets with Learnable Up- and Downsampling." In: *arXiv e-prints*, arXiv:2005.05220 (May 2020), arXiv:2005.05220. arXiv: 2005.05220 [cs.LG].

[46] Jialue Fan, Wei Xu, Ying Wu, and Yihong Gong. "Human tracking using convolutional neural networks." In: *IEEE Transactions on Neural Networks* 21.10 (2010), pp. 1610–1623.

[47] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. "Learning hierarchical features for scene labeling." In: *TPAMI* 35.8 (2013), pp. 1915–1929.

[48] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. "Deep spatial autoencoders for visuomotor learning." In: *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*. Ed. by Danica Kragic, Antonio Bicchi, and Alessandro De Luca, pp. 512–519.

[49] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana. "The SpiNNaker Project." In: *Proceedings of the IEEE* 102.5 (2014), pp. 652–665.

[50] M. S. Ganeshmurthy and G. R. Suresh. "Path planning algorithm for autonomous mobile robot in dynamic environment." In: *2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)*. 2015, pp. 1–6. DOI: 10.1109/ICSCN.2015.7219901.

[51] Marc-Oliver Gewaltig and Markus Diesmann. "NEST (NEural Simulation Tool)." In: *Scholarpedia* 2.4 (2007), p. 1430.

[52] R. Girshick, J. Donahue, and J. Malik T. Darrell. "Region-based Convolutional Networks for Accurate Object Detection and Semantic Segmentation." In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 38 (1 2015), pp. 142–158.

[53] Ross Girshick. "Fast r-cnn." In: *ICCV*. 2015, pp. 1440–1448.

[54] Georgia Gkioxari, Ross Girshick, and Jitendra Malik. "Contextual action recognition with r* cnn." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1080–1088.

[55] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. "The Reversible Residual Network: Backpropagation Without Storing Activations." In: *arXiv e-prints*, arXiv:1707.04585 (July 2017), arXiv:1707.04585. arXiv: 1707.04585 [cs.CV].

[56] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. "Speech Recognition with Deep Recurrent Neural Networks." In: *CoRR* abs/1303.5778 (2013).

[57] Kevin Gurney, Tony Prescott, and Peter Redgrave. "A computational model of action selection in the basal ganglia. II. Analysis and simulation of behaviour." In: *Biological cybernetics* 84 (July 2001), pp. 411–23. DOI: 10.1007/PL00007985.

[58] Jun Han and Claudio Moraga. "The influence of the sigmoid function parameters on the speed of backpropagation learning." In: *From Natural to Artificial Neural Computation*. Ed. by José Mira and Francisco Sandoval. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 195–201.

[59] Bharath Hariharan, Pablo Arbelaez, Ross Girshick, and Jitendra Malik. "Simultaneous Detection and Segmentation." In: *ECCV*. 2014, pp. 297–312.

[60] Hado V Hasselt. "Double Q-learning." In: *Advances in neural information processing systems*. 2010, pp. 2613–2621.

[61] Matthew J. Hausknecht and Peter Stone. "Deep Recurrent Q-Learning for Partially Observable MDPs." In: *CoRR* abs/1507.06527 (2015).

[62] Hananel Hazan, Daniel J. Saunders, Hassaan Khan, Darpan T. Sanghavi, Hava T. Siegelmann, and Robert Kozma. "BindsNET: A machine learning-oriented spiking neural networks library in Python." In: *arXiv e-prints*, arXiv:1806.01423 (June 2018), arXiv:1806.01423. arXiv: 1806.01423 [cs.NE].

[63] D. O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, June 1949. ISBN: 0-8058-4300-0.

[64] Todd Hester, Michael Quinlan, and Peter Stone. "A Real-Time Model-Based Reinforcement Learning Architecture for Robot Control." In: *CoRR* abs/1105.1749 (2011).

[65] Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. "The Goldilocks Principle: Reading Children's Books with Explicit Memory Representations." In: *CoRR* abs/1511.02301 (2015).

[66] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[67] L. Hodgkin and F. Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve." In: *The Journal of Physiology* 117.4 (1952), pp. 500–544. DOI: 10.1113/jphysiol.1952.sp004764.

[68] James B. Holliday and Ngan T.H. Le. "Follow Then Forage Exploration: Improving Asynchronous Advantage Actor Critic." In: *International Conference on Soft Computing, Artificial Intelligence and Applications (SAI 2020)* (2020), pp. 107–118.

[69] G. Huang. "Visual-Inertial Navigation: A Concise Review." In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 9572–9582. DOI: 10.1109/ICRA.2019.8793604.

[70] Dong Jin Hyun, Sangok Seok, Jongwoo Lee, and Sangbae Kim. "High speed trot-running: Implementation of a hierarchical controller using proprioceptive impedance control on the MIT Cheetah." In: *The International Journal of Robotics Research* 33.11 (2014), pp. 1417–1445. DOI: 10.1177/0278364914532150. eprint: https://doi.org/10.1177/0278364914532150. URL: https://doi.org/10.1177/0278364914532150.

[71] E. M. Izhikevich. "Which model to use for cortical spiking neurons?" In: *IEEE Transactions on Neural Networks* 15.5 (2004), pp. 1063–1070.

[72] E. Izhikevich. "Simple model of Spiking Neurons." In: *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* 14 (2003), pp. 1569–72. DOI: 10.1109/TNN.2003.820440.

[73] E. Izhikevich. *Dynamical Systems in Neuroscience: the Geometry of Excitability and Bursting*. MIT Press, 2007.

[74] E. Izhikevich. "Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling." In: *Cerebral cortex (New York, N.Y. : 1991)* 17 (Nov. 2007), pp. 2443–52. DOI: 10.1093/cercor/bhl152.

[75] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. "Deep features for text spotting." In: *European conference on computer vision*. Springer. 2014, pp. 512–528.

[76] Zeyu Jia, Lin Yang, Csaba Szepesvari, and Mengdi Wang. "Model-Based Reinforcement Learning with Value-Targeted Regression." In: *Proceedings of the 2nd Conference on Learning for Dynamics and Control*. Vol. 120. Proceedings of Machine Learning Research. The Cloud, 2020, pp. 666–686.

[77] V Craig Jordan. "Long-term adjuvant tamoxifen therapy for breast cancer." In: *Breast cancer research and treatment* 15.3 (1990), pp. 125–136.

[78] Nal Kalchbrenner and Phil Blunsom. "Recurrent Continuous Translation Models." In: Association for Computational Linguistics, 2013.

[79] Seijoon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. "Spiking-YOLO: Spiking Neural Network for Energy-Efficient Object Detection." In: *arXiv e-prints*, arXiv:1903.06530 (Mar. 2019), arXiv:1903.06530. arXiv: 1903.06530 [cs.CV].

[80] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *arXiv e-prints*, arXiv:1412.6980 (Dec. 2014), arXiv:1412.6980. arXiv: 1412.6980 [cs.LG].

[81] Brent Komer. "Biologically Inspired Adaptive Control of Quadcopter Flight." In: 2015.

[82] Vijay R Konda and John N Tsitsiklis. "Actor-critic algorithms." In: *Advances in neural information processing systems*. 2000, pp. 1008–1014.

[83] A. Krizhevsky, I. Sutskever, and G. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: 10.1145/3065386.

[84] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[85] A. Kupcsik, M. Deisenroth, Jan Peters, and G. Neumann. "Data-Efficient Generalization of Robot Skills with Contextual Policy Search." In: *AAAI*. 2013.

[86] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. "Model-Ensemble Trust-Region Policy Optimization." In: (Feb. 2018).

[87] Quoc V. Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeff Dean, and Andrew Y. Ng. "Building high-level features using large scale unsupervised learning." In: *arXiv e-prints* (Dec. 2011).

[88] Yann LeCun, D Touresky, G Hinton, and T Sejnowski. "A theoretical framework for back-propagation." In: *Proceedings of the 1988 connectionist models summer school*. CMU, Pittsburgh, Pa: Morgan Kaufmann. 1988, pp. 21–28.

[89] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. "Efficient backprop." In: *Neural networks: Tricks of the trade*. Springer, 1998, pp. 9–50.

[90] Sang-Woo Lee, Min-Oh Heo, Jiwon Kim, Jeonghee Kim, and Byoung-Tak Zhang. "Dual Memory Architectures for Fast Deep Learning of Stream Data via an Online-Incremental-Transfer Strategy." In: *arXiv e-prints*, arXiv:1506.04477 (June 2015), arXiv:1506.04477. arXiv: 1506.04477 [cs.LG].

[91] Young Lee, Yoon Lee, Hyunyong Lee, Luong Tin Phan, Hansol Kang, Uikyum Kim, Jeongmin Jeon, and Hyouk Choi. "Trajectory design and control of quadruped robot for trotting over obstacles." In: Sept. 2017, pp. 4897–4902. DOI: 10.1109/IROS.2017.8206368.

[92] M. Leshno, V. Lin, A. Pinkus, and S. Schocken. "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function." In: *Neural Networks* 6.6 (1993), pp. 861–867. DOI: https://doi.org/10.1016/S0893-6080(05)80131-5.

[93] Sergey Levine and Vladlen Koltun. "Learning Complex Neural Network Policies with Trajectory Optimization." In: *Proceedings of the 31st International Conference on Machine Learning*. 2014, pp. 829–837.

[94] Guanbin Li and Yizhou Yu. "Visual saliency based on multiscale deep features." In: *arXiv preprint arXiv:1503.08663* (2015).

[95] Jiwei Li, Minh-Thang Luong, and Dan Jurafsky. "A Hierarchical Neural Autoencoder for Paragraphs and Documents." In: *CoRR* abs/1506.01057 (2015).

[96] X. Li, W. Wang, F. Xue, and Y. Song. "Computational modeling of spiking neural network with learning rules from STDP and intrinsic plasticity." In: *Physica A Statistical Mechanics and its Applications* 491 (2018), pp. 716–728. DOI: 10.1016/j.physa.2017.08.053.

[97] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning." In: *arXiv e-prints*, arXiv:1509.02971 (Sept. 2015), arXiv:1509.02971. arXiv: 1509.02971 [cs.LG].

[98] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning." In: *arXiv e-prints*, arXiv:1509.02971 (Sept. 2015), arXiv:1509.02971. arXiv: 1509.02971 [cs.LG].

[99] Guosheng Lin, Chunhua Shen, Anton van den Hengel, and Ian Reid. "Efficient Piecewise Training of Deep Structured Models for Semantic Segmentation." In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 3194–3203.

[100] Jesus L. Lobo, Javier Del Ser, Albert Bifet, and Nikola Kasabov. "Spiking Neural Networks and online learning: An overview and perspectives." In: *Neural Networks* 121 (2020), pp. 88 –100. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.

2019.09.004. URL: http://www.sciencedirect.com/science/article/pii/S0893608019302655.

[101] J. Long, E. Shelhamer, and Tr. Darrell. "Fully convolutional networks for semantic segmentation." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[102] Thang Luong, Ilya Sutskever, Quoc V. Le, Oriol Vinyals, and Wojciech Zaremba. "Addressing the Rare Word Problem in Neural Machine Translation." In: *CoRR* abs/1410.8206 (2014).

[103] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, and Alan L. Yuille. "Deep Captioning with Multimodal Recurrent Neural Networks (m-RNN)." In: *CoRR* abs/1412.6632 (2014).

[104] T. Martinez-Marin and T. Duckett. "Fast Reinforcement Learning for Vision-guided Mobile Robots." In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. 2005, pp. 4170–4175.

[105] W. Mcculloch and W. Pitts. "A Logical Calculus of Ideas Immanent in Nervous Activity." In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 127–147.

[106] Paul A. Merolla et al. "A million spiking-neuron integrated circuit with a scalable communication network and interface." In: 345.6197 (2014), pp. 668–673.

[107] C. Meunier and I. Segev. "Playing the Devil's advocate: is the Hodgkin–Huxley model useful?" In: *Trends in Neurosciences* 25 (2002), pp. 558–563.

[108] Tomas Mikolov, Stefan Kombrink, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. "Extensions of recurrent neural network language model." In: *ICASSP*. 2011, pp. 5528–5531.

[109] Liu Mingmin, Fang Xu, Kai Jia, Qifeng Yang, and Chong Tang. "A Stable Walking Strategy of Quadruped Robot Based on Foot Trajectory Planning." In: July 2016, pp. 799–803. DOI: 10.1109/ICISCE.2016.175.

[110] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. "Human-level control through deep reinforcement learning." In: *Nature* 518.7540 (2015), pp. 529–533.

[111] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous methods for deep reinforcement learning." In: *International conference on machine learning*. 2016, pp. 1928–1937.

[112] "Model-based contextual policy search for data-efficient generalization of robot skills." In: *Artificial Intelligence* 247 (2017), pp. 415 –439.

[113] Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. "A scalable multi-core architecture with heterogeneous memory structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)." In: *arXiv e-prints*, arXiv:1708.04198 (Aug. 2017), arXiv:1708.04198. arXiv: 1708.04198 [cs.AR].

[114] I. Mordatch, N. Mishra, C. Eppner, and P. Abbeel. "Combining model-based policy search with online model learning for control of physical humanoids." In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 242–248.

[115] J. Morimoto, G. Zeglin, and C. G. Atkeson. "Minimax differential dynamic programming: application to a biped walking robot." In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*. Vol. 2. 2003, pp. 1927–1932.

[116] Jun Morimoto and Christopher Atkeson. "Nonparametric representation of an approximated Poincaré map for learning biped locomotion." In: *Auton. Robots* 27 (Aug. 2009), pp. 131–144. DOI: 10.1007/s10514-009-9133-z.

[117] Mohammadreza Mostajabi, Payman Yadollahpour, and Gregory Shakhnarovich. "Feedforward Semantic Segmentation With Zoom-Out Features." In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3376–3385.

[118] Milad Mozafari, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, and Timothée Masquelier. "SpykeTorch: Efficient Simulation of Convolutional Spiking Neural Networks With at Most One Spike per Neuron." In: *Frontiers in Neuroscience* 13 (2019), p. 625. ISSN: 1662-453X. DOI: 10.3389/fnins.2019.00625. URL: https://www.frontiersin.org/article/10.3389/fnins.2019.00625.

[119] V. Nair and G. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines." In: ICML, 2010, pp. 807–814. URL: http://dblp.uni-trier.de/db/conf/icml/icml2010.html#NairH10.

[120] Atsushi Nambu, Hironobu Tokuno, and Masahiko Takada. "Functional significance of the cortico–subthalamo–pallidal 'hyperdirect' pathway." In: *Neuroscience Research* 43.2 (2002), pp. 111 –117. ISSN: 0168-0102. DOI: https://doi.org/10.1016/S0168-0102(02)00027-5. URL: http://www.sciencedirect.com/science/article/pii/S0168010202000275.

[121] M.E. Nelson. "Electrophysiological Models In: Databasing the Brain: From Data to Knowledge." In: (2005), pp. 285–301. URL: http://nelson.beckman.illinois.edu/courses/physl317/part1/Lec3_HHsection.pdf.

[122] Trung Nguyen, Zhuoru Li, Tomi Silander, and Tze Yun Leong. "Online Feature Selection for Model-based Reinforcement Learning." In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 1. Atlanta, Georgia, USA: PMLR, 2013, pp. 498–506. URL: http://proceedings.mlr.press/v28/nguyen13.html.

[123] C. Niedzwiedz, I. Elhanany, Zhenzhen Liu, and S. Livingston. "A Consolidated Actor-Critic Model with Function Approximation for High-Dimensional POMDPs." In: *AAAI 2008Workshop for Advancement in POMDP Solvers*. 2008.

[124] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. "Learning Deconvolution Network for Semantic Segmentation." In: *The IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1520–1528.

[125] G. Papandreou, L. C. Chen, K. Murphy, and A. L. Yuille. "Weakly- and semi-supervised learning of a dcnn for semantic image segmentation." In: *ICCV*. 2015.

[126] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. "Cats and Dogs." In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2012.

[127] I. C. Paschalidis, K. Li, and R. Moazzez Estanjini. "An actor-critic method using Least Squares Temporal Difference learning." In: *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. 2009, pp. 2564–2569. DOI: 10.1109/CDC.2009.5400592.

[128] A. Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[129] Massimiliano Patacchiola and Angelo Cangelosi. "Head pose estimation in the wild using convolutional neural networks and adaptive gradient methods." In: *Pattern Recognition* 71 (2017), pp. 132–143.

[130] Jan Peters and Stefan Schaal. "Reinforcement learning of motor skills with policy gradients." In: *Neural Networks* 21.4 (2008), pp. 682 –697.

[131] M. Pfeiffer and T. Pfeil. "Deep Learning With Spiking Neurons: Opportunities and Challenges." In: *Frontiers in Neuroscience* 12 (2018), p. 774. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00774.

[132] Aske Plaat, Walter Kosters, and Mike Preuss. *Deep Model-Based Reinforcement Learning for High-Dimensional Problems, a Survey*. 2020. arXiv: 2008.05598 [cs.LG].

[133] *ROS wiki*. 2019. URL: http://wiki.ros.org/urdf.

[134] Daniel Rasmussen. "NengoDL: Combining deep learning and neuromorphic modelling methods." In: *arXiv e-prints*, arXiv:1805.11144 (May 2018), arXiv:1805.11144. arXiv: 1805.11144 [cs.NE].

[135] Daniel Rasmussen and C. Eliasmith. "A neural model of hierarchical reinforcement learning." In: *PLoS ONE* 12 (2014).

[136] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." In: *CoRR* abs/1506.01497 (2015).

[137] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation." In: *arXiv e-prints*, arXiv:1505.04597 (May 2015), arXiv:1505.04597. arXiv: 1505.04597 [cs.CV].

[138] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 0033-295X. DOI: 10.1037/h0042519. URL: http://dx.doi.org/10.1037/h0042519.

[139] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, and Michael Pfeiffer. "Theory and Tools for the Conversion of Analog to Spiking Convolutional Neural Networks." In: *arXiv e-prints*, arXiv:1612.04052 (Dec. 2016), arXiv:1612.04052. arXiv: 1612.04052 [stat.ML].

[140] David E Rumelhart. "The architecture of mind: A connectionist approach." In: *Mind readings* (1998), pp. 207–238.

[141] T. P. Runarsson and S. M. Lucas. "Imitating play from game trajectories: Temporal difference learning versus preference learning." In: *2012 IEEE Conference on Computational Intelligence and Games (CIG)*. 2012, pp. 79–82. DOI: 10.1109/CIG.2012.6374141.

[142] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. "Imagenet large scale visual recognition challenge." In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[143] Doyen Sahoo, Quang Pham, Jing Lu, and Steven C. H. Hoi. "Online Deep Learning: Learning Deep Neural Networks on the Fly." In: *arXiv e-prints*, arXiv:1711.03705 (Nov. 2017), arXiv:1711.03705. arXiv: 1711.03705 [cs.LG].

[144] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. "Trust Region Policy Optimization." In: *arXiv e-prints* (Feb. 2015).

[145] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms." In: *arXiv e-prints*, arXiv:1707.06347 (July 2017), arXiv:1707.06347. arXiv: 1707.06347 [cs.LG].

[146] M. R. Shaker, Shigang Yue, and T. Duckett. "Vision-based reinforcement learning using approximate policy iteration." In: *2009 International Conference on Advanced Robotics*. 2009, pp. 1–6.

[147] Sumit Bam Shrestha and Garrick Orchard. "SLAYER: Spike Layer Error Reassignment in Time." In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., 2018, pp. 1419–1428. URL: http://papers.nips.cc/paper/7415-slayer-spike-layer-error-reassignment-in-time.pdf.

[148] Bing Shuai, Gang Wang, Zhen Zuo, Bing Wang, and Lifan Zhao. "Integrating parametric and non-parametric models for scene labeling." In: *CVPR*. 2015, pp. 4249–4258.

[149] B Siddoway, H. Hou, and H. Xia. "Molecular mechanisms of homeostatic synaptic downscaling." In: *Neuropharmacology* 78 (2014). Homeostatic Synaptic Plasticity, pp. 38 –44. ISSN: 0028-3908. DOI: https://doi.org/10.1016/j.neuropharm.2013.07.009. URL: http://www.sciencedirect.com/science/article/pii/S0028390813003262.

[150] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556* (2014).

[151] S. Song, K. Miller, and L. Abbott. "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity." In: *Nature Neuroscience* 3.9 (2000), pp. 919–926. ISSN: 10976256. DOI: 10.1038/78829.

[152] S. Sonoda and N. Murata. "Neural Network with Unbounded Activation Functions is Universal Approximator." In: *arXiv e-prints*, arXiv:1505.03654 (2015), arXiv:1505.03654. arXiv: 1505.03654 [cs.NE].

[153] M. Stemmler and C. Koch. "How voltage-dependent conductances can adapt to maximize the information encoded by neuronal firing rate." In: *Nature neuroscience* 2 (July 1999), pp. 521–7. DOI: 10.1038/9173.

[154] Terrence Stewart, Xuan Choo, and Chris Eliasmith. "Dynamic Behaviour of a Spiking Model of Action Selection in the Basal Ganglia." In: Jan. 2010.

[155] A. Strassberg and L. DeFelice. "Limitations of the Hodgkin-Huxley Formalism: Effects of Single Channel Kinetics on Transmembrane Voltage Dynamics." In: *Neural Computation* 5 (1993), pp. 843–855.

[156] Richard S Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation." In: *Advances in Neural Information Processing Systems 12*. 2000, pp. 1057–1063.

[157] Sebastian Thrun. "Simultaneous Localization and Mapping." In: *Robotics and Cognitive Approaches to Spatial Mapping*. Ed. by Margaret E. Jefferies and Wai-Kiang Yeap. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 13–41.

[158] Alexander Toshev and Christian Szegedy. "Deeppose: Human pose estimation via deep neural networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1653–1660.

[159] Anne-Marie Tousch, Stéphane Herbin, and Jean-Yves Audibert. "Semantic hierarchies for image annotation: A survey." In: *Pattern Recognition* 45.1 (2012), pp. 333–345.

[160] Lijun Wang, Huchuan Lu, Xiang Ruan, and Ming-Hsuan Yang. "Deep networks for saliency detection via local estimation and global search." In: *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE. 2015, pp. 3183–3192.

[161] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. "Benchmarking Model-Based Reinforcement Learning." In: *CoRR* abs/1907.02057 (2019).

[162] Z. Wang and W. Wong. "Key role of voltage-dependent properties of synaptic currents in robust network synchronization." In: *Neural networks : the official journal of the International Neural Network Society* 43C (Feb. 2013), pp. 55–62. DOI: 10.1016/j.neunet.2013.01.024.

[163] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. "Dueling network architectures for deep reinforcement learning." In: *arXiv preprint arXiv:1511.06581* (2015).

[164] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[165] Aaron Wilson, Alan Fern, and Prasad Tadepalli. "Using Trajectory Data to Improve Bayesian Optimization for Reinforcement Learning." In: *Journal of Machine Learning Research* 15.8 (2014), pp. 253–282. URL: http://jmlr.org/papers/v15/wilson14a.html.

[166] C. Wirth and J. Fürnkranz. "On Learning From Game Annotations." In: *IEEE Transactions on Computational Intelligence and AI in Games* 7.3 (2015), pp. 304–316. DOI: 10.1109/TCIAIG.2014.2332442.

[167] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. "Empirical Evaluation of Rectified Activations in Convolutional Network." In: *arXiv e-prints*, arXiv:1505.00853 (May 2015), arXiv:1505.00853. arXiv: 1505.00853 [cs.LG].

[168] Hailiang Xu and Feng Su. "Robust seed localization and growing with deep convolutional features for scene text detection." In: *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*. ACM. 2015, pp. 387–394.

[169] Dong Yu, Wayne Xiong, Jasha Droppo, Andreas Stolcke, Guoli Ye, Jinyu Li, and Geoffrey Zweig. "Deep Convolutional Neural Networks with Layer-Wise Context Expansion and Attention." In: *Interspeech*. 2016, pp. 17–21.

[170] Davide Zambrano and Sander M. Bohte. "Fast and Efficient Asynchronous Neural Computation with Adapting Spiking Neural Networks." In: *arXiv e-prints*, arXiv:1609.02053 (Sept. 2016), arXiv:1609.02053. arXiv: 1609.02053 [cs.NE].

[171] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks." In: *ECCV*. 2014, pp. 818–833.

[172] Xuanqi Zeng, Songyuan Zhang, Hongji Zhang, Xu Li, Haitao Zhou, and Yili Fu. "Leg Trajectory Planning for Quadruped Robots with High-Speed Trot Gait." In: *Applied Sciences* 9 (Apr. 2019), p. 1508. DOI: 10.3390/app9071508.

[173] Daochen Zha, Kwei-Herng Lai, Kaixiong Zhou, and Xia Hu. "Experience replay optimization." In: *arXiv preprint arXiv:1906.08387* (2019).

[174] Jing Zhang, Wanqing Li, Philip O Ogunbona, Pichao Wang, and Chang Tang. "RGB-D-based action recognition datasets: A survey." In: *Pattern Recognition* 60 (2016), pp. 86–105.

[175] Guanyu Zhou, Kihyuk Sohn, and Honglak Lee. "Online Incremental Feature Learning with Denoising Autoencoders." In: *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Neil D. Lawrence and Mark Girolami. Vol. 22. Proceedings of Machine Learning Research. La Palma, Canary Islands: PMLR, 2012, pp. 1453–1461. URL: http://proceedings.mlr.press/v22/zhou12b.html.

[176] *pigpio library*. 2020. URL: http://abyz.me.uk/rpi/pigpio/pigpiod.html.

[177] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-learning." In: *arXiv e-prints*, arXiv:1509.06461 (Sept. 2015), arXiv:1509.06461. arXiv: 1509.06461 [cs.LG].

[178] Muhammed Arif Şen, Veli Bakırcıoğlu, and Mete Kalyoncu. "Inverse Kinematic Analysis Of A Quadruped Robot." In: *International Journal of Scientific Technology Research* 6 (Oct. 2017).

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

`https://bitbucket.org/amiede/classicthesis/`