UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

THE CONTEXT-AWARE LEARNING MODEL

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

JOOHEE SUH
Norman, Oklahoma
2017

THE CONTEXT-AWARE LEARNING MODEL

A DISSERTATION APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY

Dr. Dean F. Hougen, Chair

Dr. Ingo Schlupp

Dr. Sridhar Radhakrishnan

Dr. S. Lakshmivarahan

Dr. John Antonio

DEDICATION

to

My dog, Heemang

A Korean Female Dog Named Heemang

For

Being with me under all circumstances since born in 2006.

Acknowledgments

I would like to send my deep gratitude to my Ph.D. advisor, Professor Dean F. Hougen. I could learn from him how to do research in a more precise way and how to organize extensive information related to the research work in a professional and efficient way. Especially, I could learn what kind of preliminary research steps should be preceded in a very concrete way; and thus I could broaden my perspectives in data handling, research analysis, and technical writing. All of his feedback (not only on my research work but also on my homeworks from his classes that I took) are valuable for me as I could realize the followings: (1) what I should improve, (2) what I missed, and (3) what I did well. Moreover, I appreciate his consistent encouragement and patience whenever the visible research results were not as I expected or research progress was not moving as fast as I planned.

I would like to send my appreciation to Professor Chong-Woo Woo who was my Master advisor in Seoul, South Korea. The first time I started to have my interests in artificial intelligence (AI) and smart robotics was when I took his undergraduate AI course at Kookmin University in 2006. I could step into the research world since I wrote my first conference paper with him under his great advising. Also, he was the person who strongly recommended for me to study abroad especially in the USA and thus I could take my courage to move to the USA.

I would like to send my appreciation to all the professors who gave me

research assistant positions during my doctoral career so that I could broaden my academic experiences and personal connections. I could survive financially and thus could finalize this dissertation thanks to Dr. Andrew H. Fagg, Dr. Joseph P. Havlicek, Dr. Ronald D. Barnes, and Dr. Chris Weaver.

Finally, I would like to show my sincere appreciation to all the Professors who are in my Ph.D. committee: Dr. Dean F. Hougen, Dr. Ingo Schlupp, Dr. Sridhar Radhakrishnan, Dr. S. Lakshmivarahan, and Dr. John Antonio. I could finally accomplish my doctoral degree with the great support from my committee. Thank you.

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

The ultimate goal of this research is to build a novel, generalized, arbitrary-depth, neural controller that performs reward- and experience-based neuromodulatory learning, which is online, bootstrapping, interactive, incremental, and dynamic. Autonomous agents, such as robots, maybe able to adapt to uncertain environments if they use reward-based, interactive learning. Unfortunately, typical reward-based models are based on discrete state and action spaces whereas many interesting applications contain continuous spaces. This suggests the use of an artificial neural controller with continuous weights. Adapting the neuromodulatory features of biological brains into a robot controller plays an important role in building more biological robots; however, a biologically feasible learning model does not necessarily promote increased learning efficiency or optimizing the neural networks in a generalized way. For these reasons, this research introduces the Context-Aware Learning Model (CALM) and four different learning algorithms that operate within this model, all of which use logistic regression backpropagation and hyperbolic, reward-based learning. This research introduces a novel way of combining reward- and experience-based learning with an arbitrary-depth artificial neural network and shows how specific behavioral neurobiological features are applied in building a novel neuromodulatory learning mechanism. CALM is evaluated with five metrics on six synthetic data sets and shows promising performances.

Chapter 1

Introduction

This chapter provides the overall motivation for the research in this dissertation, lists and explains the research questions that drive the approach, introduces concepts and characteristics of the CALM, and finally outlines the rest of this dissertation.

1.1 Motivation

There have been increasing numbers of investigations on building novel bio-inspired learning models, especially in the neurorobotics area. In the neurorobotics area, most robot behavioral control research focuses on adapting biological neuromodulatory processes into robot behavior decisions following biologically demonstrated concepts of vertebrate brains [15] [50] [23]. These learning models show successful performances application to specific domain problems and open more possibilities of building more biologically intelligent and practical robots. Some of this research focuses on simulating brain dynamics that mimic certain parts of vertebrate brains. However, most neurorobotics research has not been as generalized as the typical machine learning approach and usually focused on how to graft neurobiological features onto a computational system with domain-specific data and fixed neural framework. In this regard, this dissertation introduces a novel, generalized, arbitrary-depth, neural network inspired by four features of behavioral neurobiology: (1) combination-sensitive

neurons, (2) recurrent inhibition, (3) appetitive learning with serotonergic neuromodulation, and (4) aversive learning with dopaminergic neuromodulation.

There are limitations for an agent to adapt to an unknown or changing environment if the agent is only designed to use a supervised learning model. A supervised learning model, such as a typical neural network that uses an optimization method such as logistic regression backpropagation (LRB) or least mean square optimization (LMS or delta rule), is based on a fixed set of training data with a static optimization method [11] [43]. Therefore, if, after deployment, there is new input data that was not covered by the original training data or if the desired output changes in a dynamic environment, an agent with a supervised learning model might have some limitations on its decision making since the new information had not been trained before exploring the world. In other words, a supervised learning is not designed to be applied to dynamic or online learning problems since it depends on being given correct answers about a static world. However, it can be a powerful learning method in handling complex and sophisticated classification problems under the two conditions: (1) it has appropriate training data which is sufficient to cover the given input problem space and (2) it is applied to static data. CALM embraces the sound computational process of logistic regression optimization (as described in Section 3.2) and thus its benefits CALM adapts this supervised optimization method to dynamically changing environments without target outputs by using rewards.

Reward-based learning exploits feedback information from an environment for its decision making. The reward-based learning types can be classified depending on the way they use the feedback information. If the learning system has a state transition model and predicts which would be the next state

based on a reward value, it is generally called *reinforcement learning* [57] [64]. On the other hand, if a model directly utilizes reward for updating its neural weights, it is generally called *reward-based neural learning*. Both types have been investigated and applied to various learning domains and have shown good performance [30] [11]. However, the both learning types may not be designed for a generalized optimization process. CALM introduces how reward information is applied into a generalized, arbitrary-depth, neural optimization process and shows its performance based on several sets of generalized synthetic data.

In summary, this research is motivated by three different areas: (1) bio-inspired learning, especially based on behavioral neurobiology, (2) supervised learning, and (3) reward-based learning. Each area has its own learning benefits and CALM aims to be a novel learning model which is able to have the benefits of each learning method. The details of CALM are fully described in Chapter 5.

1.2 Research Questions

The motivation and enthusiasm for this dissertation comes from two perspectives: (1) general machine learning and (2) bio-inspired learning. Based on this, the research questions are described from each perspective. Note that the following questions are not intended to cover the detailed topics of this research but are intended to give general cues for outlining this research.

First, in terms of general machine learning: Is there a generalized, arbitrary-depth, neural learning model that provides the following abilities simultaneously? (1) is able to recognize the similarities and differences between contexts based

on rewards from its environment, (2) can learn without a pre-defined world model or pre-structured knowledge-base, (3) can learn from both the current situation and past learning experiences, and (4) is able to adapt to a dynamically changed environment. This multi-part question represents a big motivation for CALM followed by sub questions: Will using memorized experiences improve its learning ability? And if so, what is an appropriate computational process for that and is the model generalizable with sound mathematical derivations?

Second, in from the bio-inspired learning perspective, there is one more important question in addition to the above research questions: Is there a generalized, bio-inspired, model that is able to demonstrate benefits compared to non-bio-inspired learning model?

With the above conceptual questions, the ultimate goal of this research is to build a novel, bio-inspired, arbitrary-depth, neural learning model that is online, bootstrapping, interactive, incremental, and dynamic. The intention is to be able to apply CALM to robotics as well as to non-robotics domains.

1.3 The Overview of the Context-Aware Learning Model (CALM)

This dissertation introduces the Context-Aware Learning Model (CALM). This dissertation also introduces four implementations of CALM: (1) CALM reward-based Logistic Regression Backpropagation (CALM-rLRB), (2) CALM experience-based Logistic Regression Backpropagation (CALM-eLRB), (3) CALM

experience-powered Logistic Regression Backpropagation (CALM-epLRB), and (4) CALM neuromodulatory experience-powered Logistic Regression Backpropagation (CALM-nepLRB). The first three algorithms, CALM-rLRB, CALM-eLRB, and CALM-epLRB, use the novel, arbitrary-depth, neural learning model proposed herein and the last one, CALM-nepLRB, adds additional novel, bio-inspired mechanisms. CALM-nepLRB is inspired by four features of neurobiology: (1) combination-sensitive neurons, (2) recurrent inhibition, (3) appetitive learning with serotonergic neuromodulation, and (4) aversive learning with dopaminergic neuromodulation.

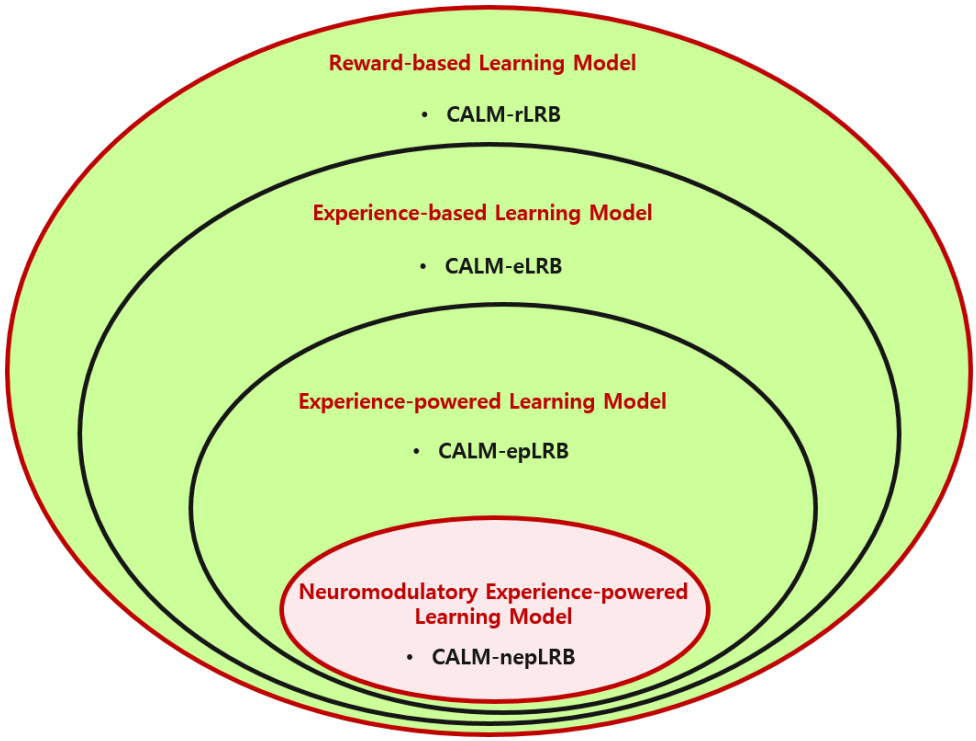


Figure 1.1: CALM Algorithm Venn Diagram

The Figure 1.1 conceptually specifies how four different CALM algorithms are classified in the form of Venn diagram. All of the algorithms are reward-based, so this is the outer ring within which they all belong. CALM-rLRB is

the simplest of these algorithms, using only reward as the basis of its learning, so it belongs only in the outer ring. Moving in one ring, the concept of an experience is introduced. An experience is synthetic information which includes a context, a corresponding output, and a corresponding reward. Algorithms that are experience-based use these experiences to optimize its artificial neural connections. CALM-eLRB falls into this category. Moving in another ring, we find experience-powered algorithms. An experience-powered algorithm is one that uses extended experiences, which utilizes a past successful neural connection as well as past experiences; CALM-epLRB falls into the category of experience-powered learning algorithms. In the inner ring, neuromodulation is added to provide additional behavioral and learning features. Embracing all of these features, CALM-nepLRB, is most novel and sophisticated of the algorithms proposed herein. The Table 1.1 clarifies the different learning features of each algorithm. The detail principles and specifications of each algorithms are described in Chapter 5 and the performances comparison are shown and discussed in Chapter 6. Note that, in the Table 1.1, 'CALM' is omitted in each algorithm name in order to reduce the table size.

1.4 CALM Concepts and Algorithms

Based on the concept of CALM, a brief explanation of each algorithm and definitions of terminologies used in this dissertation are described as follows.

Context awareness originates from ubiquitous computing [65] [66].

> In ubiquitous computing, "context is any information about the circumstances, objects, or conditions by which a user is surrounded that is considered relevant to the interaction between the user and

the ubiquitous computing environment [49] [48]."

> "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves [1] [10]."

> "Context-Aware Computing is the use of context to provide relevant information and/or services to the user, where relevancy depends on the particular task of the user. A System is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task [1] [10]."

Context in CALM is defined as any combined, compressed, or encoded information which can be reasonably used as input to an artificial neural network. Therefore each context can be considered to be an input vector for the network.

Context-awareness, then, is defined in this dissertation as learning an appropriate action to take in a given context through inference from a well-structured knowledge based.

CALM-rLRB uses reward-based learning through logistic regression back-propagation using feedback from its environment.

CALM-eLRB uses a knowledge base of experiences to ensure that adjustments to its neural weights to accommodate each new experience do not obscure what it has already learned. An *experience* refers to integrated synthetic information indicating (1) a given context, (2) the output the system selected in response to the context, (3) the feedback value received as a result of the selection made, and (4) neural strength between the context and the selected output.

CALM-epLRB uses an additional mechanism called the selective power

update to adjust the current weights of the network based on weights saved in the knowledge base from a prior extended experience.

CALM-nepLRB adds additional neurobiological features: (1) combination-sensitive neurons, (2) recurrent inhibition, (3) appetitive learning with serotonergic neuromodulation, and (4) aversive learning with dopaminergic neuromodulation.

1.5 CALM Characteristics

In this section, terms used to describe CALM characteristics are defined and the algorithms are mapped to the characteristics.

ONLINE Online indicates that the learning model continually takes in new data as it learns, rather than needing its entire data set to be provided before learning begins. Online learning helps a learning model to be adaptive in an uncertain environment. All CALM algorithms are online learning models that explore environments by getting new input data. For example, if selecting *output 1* gets reward in *context A*, CALM considers the situation as a positive experience and memorizes the experience to exploit them in next learning iteration. On the other hand, if selecting *output 1* gets negative feedback in *context A*, it keeps exploring the environment without memorizing it since it is considered as an error.

INCREMENTAL Incremental learning is a special type of online learning. Incremental learning indicates that the amount of data used in a single learning step increases over learning steps by including newly added data. In other words, while online learning might update its learned representation (e.g., the weights

of its neural network) at time t based only on new data received at time t, incremental learning updates its learned representation based on all accumulated data up to and including that from time t. Therefore, all incremental learning is online learning but not vise versa. CALM-eLRB, CALM-epLRB, and CALM-nepLRB processes different number of input data at each learning step. For example, the number of empirical experiences are incrementally building up over learning steps and thus the increased number of experiences are applied to optimization process in each learning step. Note that only CALM-rLRB is not incremental learning since it utilizes only current input in its learning process, which is not experience-based.

ARBITRARY DEPTH Arbitrary depth means that a learning model can have more or fewer layers in its neural network. CALM supports learning that can be considered shallow, moderate, or deep. Note that, in this dissertation, the number of layers of an ANN refers to the total number of layers including input, output, and hidden layers. For example, if there is a neural network with having no hidden layer, it is considered as 2-layered neural network. In this regard, arbitrary-depth neural learning means that an artificial neural network has at least greater than or equals to four layers in optimizing its neural connections. CALM shows its learning benefits depending on the depth; therefore, it is notable that CALM does not have to use a fixed depth of neural network since it is general neural network. The number of layers of an artificial neural network in CALM depends on a system designer's intention. The experimental results of shallow, moderate, and deep learning are addressed in Chapter 6.

BOOTSTRAPPING Bootstrapping means that the model learns an environment without needing prior knowledge or a pre-defined world model. In

other words, bootstrapping refers that a learning model starts from scratch to understand an environment while adaptive learning means it exploits 'try and errors' to get better understanding over time. In a broader sense, CALM has several assumptions for learning from zero knowledge about any environment it might encounter: (1) there is exactly one correct response for each context, (2) responses are limited to a fixed number of discrete alternatives, (3) contexts can be described by vectors, and (4) similar contexts can be determined by looking at the Euclidean distance between context vectors. If any (or all) of these assumptions are violated, CALM will not learn appropriately. Based on this assumptions, all of CALM algorithms are bootstrapping and especially CALM-eLRB, CALM-epLRB, and CALM-nepLRB build up its own knowledge base and exploits them over learning process.

INTERACTIVE Interactive means that the learning model uses feedback information which is either rewards (positive feedback) or punishments (negative feedback) from a responsive process (e.g., sensing the world or from a trainer).

DYNAMIC Dynamic means that a learning model can handle environments that change during the learning process. When learned associations between input and output (or rewards) are changed, a dynamic model should be able to adapt to the changed situations. In this case, CALM-rLRB and CALM-nepLRB are dynamic learning algorithms in that they can adapt to newly changed environment. Briefly, CALM-rLRB always uses only current context and thus it is dynamic. CLAM-nepLRB can change associations based on negative feedback from the environment. Therefore, CALM-nepLRB is able to learn not only environments which has not encountered before but also changed environments.

Index	Learning Features	-rLRB	-eLRB	-epLRB	-nepLRB
1	Online	x	x	x	x
2	Arbitrary Depth	x	x	x	x
3	Bootstrapping	x	x	x	x
4	Interactive	x	x	x	x
5	Incremental		x	x	x
6	Dynamic	x			x

Table 1.1: CALM Characteristics

1.6 Organization of the Dissertation

The rest of this dissertation is organized as follows. Essential neuroscience, which are necessary knowledge for building a machine brain algorithm, CALM-nepLRB, and understanding the related preceding research works in bio-inspired learning area, are explained in Chapter 2. The overview of artificial neural networks including a reward-based neural model are briefly covered and the logistic regression neural networks which are the key background knowledge for CALM are fully described in Chapter 3. It also provides sound mathematical derivations for the learning methods, which supports the computational solidity of CALM algorithms. Related research works regarding to CALM with three sections: (1) reward-based learning, (2) neurorobotics learning, and (3) context-based robot learning are introduced in Chapter 4. The overview of CALM is introduced and the details for the four different learning algorithms: CALM-rLRB, CALM-eLRB, CALM-epLRB, and CALM-nepLRB are explained in Chapter 5. In the section, we can see the differences among the four algorithms and how CALM-nepLRB is incrementally designed from the basic algorithm CALM-rLRB. The experimental setup for evaluating CALM are described and its promising results based on several synthetic data sets are showed in Chapter 6. Several discussions regarding to build each learning model of CALM are described in

Chapter 7 and the conclusion and the contributions of this research work are explained in Chapter 8. Finally, future work of this dissertation is described in Chapter 9.

Chapter 2

Essential Neuroscience

This chapter provides basic knowledge of neuroscience which are essential to build the CALM. Especially, this chapter focuses on describing the definition of neurons and neural system, two types of neural communication, several basic neurotransmitters, and four features of animal behavioral neurobiology.

2.1 A Neuron

Nervous system comprise nerve cells and supporting cells. Neurons refer to nerve cells. Supporting cells are also called the neuroglia cells or glia cells. The big difference from the two types of cell is that neurons have good structures for electrical signaling but glia cells are not able to perform electrical signaling by themselves. A neuron consists of 5 components: (1) cell body(soma), (2) axon hillock, (3) axon, (4) axon terminal(s), and (5) dendrite(s). Each neuron has one soma and one axon; but the axon can branch out as many as it needs so the neuron can have several axon terminals which can make contacts to the other neurons. Axon hillock is the initial point of sending electrical signal from the cell body. Dendrites can be considered as dendritic tree which receives and integrates information from the other neurons. Therefore, in a computational aspect, neurons are units which can takes and transfer information from the other different neurons. Dendritic tree functions as taking input from the other neurons and axon terminals perform sending output to the others, which is the

basic system for neural communication [13] [15].

2.2 The Electrical Signals of Neurons

Electric signals are generated by voltage difference between inside and outside membrane of cells. Potential indicates the voltage difference, which is also called activation level. There are 3 types of potential: (1) equilibrium potential, (2) resting potential, and (3) action potential. Equilibrium potential indicates the static voltage difference value when there are no ionic movements or flows across the cell. Each different type of ions takes different value of equilibrium potential so a cell's equilibrium potential depends on by what kind of ions the cell is surrounded. Resting potential refers to the static voltage difference value when there are stable ionic movements in and out of the cell, meaning the total potential is stable although different ions keep moving across the cell [13].

Regarding to dynamics of the potential, there are three states that a cell can have: (1) depolarization, (2) hyperpolarization, (3) repolarization. Depolarization refers to the state where voltage value of inside cell is less negative than the resting potential. Hyperpolarization means the value of inside cell is more negative than the resting potential. And Repolarization refers to the state where the cell recovers its potential toward the original resting potential after having hyperpolarization. In depolarization, if the potential goes positive over a certain threshold, a cell causes sharp electrical discharge and that is the action potential which is so-called a spike or pulse. When a neuron shows spike, it is usually said that "neuron is fired" [13] [16].

More specifically, an action potential occurs with having a period of four successive phases of a cell: (1) depolarization, (2) action potential, (3) hyperpolarization, and (4) repolarization. The action potential starts from depolarization; and if it goes over a certain threshold, it shows electrical discharge then it is followed by hyperpolarization and repolarization. In this way, a cell generates electrical signals through the active voltage difference between in and out of its membrane and this signals are used to transmit/communicate information between neurons [13].

2.3 The Communications of Neurons

The key role of the neural communication is synaptic transmissions among neurons. For the simplicity, if we focus on just two neurons, the one neuron sending signals is called presynaptic neuron and the other neuron receiving the signals is called postsynaptic neuron. The signal transmissions among the two neurons are performed at synaptic contact point(s). A synaptic contact point is the specific area between a presynaptic axon terminal and a postsynaptic dendrite. The specialized synaptic contacting process at a synaptic contact point is called a synapse. In other words, a synaptic contact point is where the information is transmitted and a synapse refers to the process of transmitting the information; so a synapse is also referred to as a synaptic contact [13].

Generally, the synapses are divided into two types: (1) electrical synapses and (2) chemical synapses [25]. Electrical synaptic contacts happen when neurons communicate through electrical synapses and chemical synaptic contacts occur when neurons communicate through chemical synapses. For the electrical

synaptic contacts, action potential, which is electrical event generated from presynaptic axon hillock, is flew to the presynaptic axon terminals; then it moves on to the postsynaptic dendrites through gap junctions which are the specialized proteins on dendrites. For the chemical synaptic contacts, the most communications are made in synaptic cleft which is extracellular space between a presynaptic axon terminal and a postsynaptic dendrite. When an action potential is arrived at presynaptic axon terminal, unlike the electrical synapses, neurotransmitter molecules are diffused from the terminal into the synaptic cleft; then the molecules binds to receptors at the postsynaptic specialization. In other words, chemical synapse needs neurotransmitters from a presynaptic neuron and it needs to bind a certain type of receptors which stay at postsynaptic dendrites correspond to the neurotransmitters. In this way, a presynaptic neurons can pass the signal, an action potential, to a postsynaptic neuron through electrical or chemical synapses [13].

After receiving the action potential, there are two types of response that the postsynaptic neuron can show: (1) Excitatory Postsynaptic Potential (EPSP) and (2) Inhibitory Postsynaptic Potential (IPSP). If a postsynaptic neuron shows EPSP, it means the neuron probably increases its own action potential. Otherwise, if a neuron shows IPSP, it means the neuron probably decreases its own action potential due to the effects of the received signals from the presynaptic neuron; therefore, IPSP acts as blocking the signal transmission from the presynaptic neuron and EPSP serves as enhancing the signal transmission [13].

2.4 Neurotransmitters

There are several well-known neurotransmitters related to EPSP or IPSP. Glutamate is the standard neurotransmitter showing EPSP and it takes effects when binding to two types of receptor: AMPA and NMDA. On the other hand GABA is the typical example of IPSP type neurotransmitter and it binds to GABA receptor. Serotonin also shows EPSP with the 5-HT receptor [13]. Dopamine is kind of special type of neurotransmitter because it triggers EPSP when binding to the receptors: INDR and D1-like receptors, but causes IPSP when binding to D2-like receptors [4] [33].

2.5 Neural Systems

It is already very well known that all neural systems are not same. Neural systems can be divided into several types by functionally or anatomically. By functionality, there are three neural systems: (1) sensory system, (2) motor system, and (3) associational system. Sensory system gets information from the environment. Motor system shows appropriate output responding to the sensory input in the form of actions or behaviors. Associational system make a connection between two neural systems and perform complex functions. By anatomically, neural systems are divided into central nervous system (CNS) and peripheral nervous system (PNS). CNS comprises the brain and spinal cord; and PNS includes the sensory division and motor division [13] [15].

2.6 Animal Behavioral Neurobiology

This section introduces essential background of each animal behavioral neurobiological concept: (1) combination-sensitive neurons, (2) recurrent inhibition, (3) appetitive learning with serotonergic neuromodulation, and (4) aversive learning with dopaminergic neuromodulation.

Combination-Sensitive Neurons Some animals have combination-sensitive neurons such as bat, owl, and electric eel. A combination-sensitive neuron means that it shows its responses when at least 2 different types of input neuron are given to them. The first example of the combination-sensitive neurons is the AC (Auditory Cortex) area of a bat brain which includes briefly two types of combination-sensitive neurons in two sub areas: FM (Frequency Modulated)-FM and CF-CF (Constant-Frequency) area [60] [47]. A bat performs echolocation behavior in order to find the location of a target object. In a bat's echolocation behavior, the bat generates two types of information such as relative velocity and distance information from different type of combination sensitive sensory neurons. The combination-sensitive neurons for gathering distance information are distributed in FM-FM area in AC. Those neurons are selectively responsive to the particular combination of pulse-echo time delay. In other words, some neurons are active when echo sound takes long time to arrive while the other neurons are only active when the echo sound comes in a short time based on a same pulse signal. In this way, the neurons in FM-FM area takes charge in generating distance information with the FM pulse and FM echo combination-sensitive neurons. On the other hand, CF-CF area includes the combination-sensitive neurons for encoding relative velocity information. A bat uses Doppler shift frequency by checking the difference between pulse CF and echo CF values. In

other words, the neurons in CF-CF area of a bat brain shows their different responses with the different combination of pulse CF and echo CF values. In this way, a bat can generate frequency map and distance map from AC where each neuron in each different place of the brain responses to each desired pulse and echo time delay.

The second example is ICX (external nucleus of the inferior colliculus) area of an owl's brain [37] [29] [18] [20]. An owl performs sound localization behavior to find a target location. In an owl's sound localization, the owl needs two types of information such as ITD (interaural Time Difference) and IID (Interaural Intensity Difference). From a sound source, neurons in Nucleus magnocellularis encodes azimuth information where the neurons shows different responses to different value of ITD between left and right ear. On the other hand, neurons in Nucleus angularis encodes elevation information by checking the sound level differences from left to right ear. With these two types of information, each neuron in ICX shows its activity based on different combination of ITD and IID. In this way, the owl can localize a target object's location form a sound source with the combination-sensitive neurons in ICX.

The third example of the combination-sensitive neurons is TS (Torus Semicircularis) in the brain of a weakly electric fish. A weakly electric fish shows JAR (Jamming Avoidance Response) behavior which is literally to avoid electric jamming between same species. It is known that electric fishes generates electric signals in order to communicate or detect objects and they have their own level of electric frequency so that they can avoid their interference. In an electric fish's JAR behavior, two type of sensory information are needed such as AM (Amplitude Modulation) and PM (Phase Modulation) [68] [6]. AM refers

to the intensity difference of EOD (Electric Organ Discharge) between own body and the other's body and it is received from the P type electroreceptor of an electric fish. This AM information can be the source for an electric fish to make a decision whether to increase or decrease its EOD. PM refers the phase difference between itself and the other and it is given from the T type electroreceptor of an electric fish. This PM information is necessary to know whose EOD is first recognized. With those two types of information, an weakly electric fish can adjust its frequency discharge by calculating the frequency differences based on the combination-sensitive neurons in TS. For example, if its own EOD occurs advanced compared to the other one and AM decreases then some neurons in TS are activated which indicates the other one has higher frequency which is denoted as +Df; in this case, the electric fish decrease its EOD.

Recurrent Inhibition CALM-nepLRB also has unique characteristic compared to the other three CALM algorithms, which is inspired escape behavior of a crayfish. It is known that a crayfish performs tail-flip escape behavior when having three types of stimulation from (1) LGs (Lateral Giant neurons), (2) MGs (Medial Giant neurons), and (3) non-giant neurons [41] [67]. First of all, LGs are activated when the caudal tactile is stimulated and the activation causes upward tail-flip escape, which is called LG-mediated tail flip. On the other hand, MGs shows responses when the rostral tactile is stimulated and it triggers backward tail-flip escape, which is called MG-mediated tail flip. In this regard, it is known that the activation of those giant neurons are necessary for the escape behavior. In here, there is interesting question: what if the caudal or rostral tactile is stimulated continuously?. The answer is that a crayfish will not show the corresponding reactions continuously like an electrically energized robot. LGs

and MGs shows recurrent inhibition which prevents its activation for a certain amount of the time when the stimulations are occurs successively. In other words, a neural circuit of a crayfish is designed that if the same stimulation occurs continuously in a certain duration then inhibition process for the corresponding neurons arises with releasing GABA neurotransmitter which causes IPSP. In this way, a crayfish can avoid repeating the same behavior all the time.

Appetitive Learning with Serotonin vs Aversive Learning with Dopamine There are preceding researches on effects of some neuromodulators regarding to appetitive learning and aversive learning from natural animals. CALM-nepLRB is inspired from the neurobiological behavioral learning processes with specific neuromodulators of moth, honeybee, and drosophila. For moth, it is known that the neuromodulator, serotonin, is released in greater amounts in the antennal lobes (ALs) at the specific times when the Datura flower opens. This implies two things: (1) the moths exhibit more response when their host plants open the flowers and (2) the increasing release of serotonin make the AL neurons more sensitive [2]. Also, serotonin gives the moths a periodic sensory cues for their olfactory coding [35]. Therefore, the release of serotonin plays a role in triggering appetitive behavior for moth and it is natually correleated to the environmental contexts such as flowering.

On the other hand, it is known that the release of dopamine in antennal lobe cell bodies is important to the aversive learning for honeybees and drosophila [31] [63] [40]. Specifically, the effects of octopamine and dopamine on the olfactory responses of drosophila were tested early. In Schwaerzel et al.'s experiments, they used sugar learning as an appetitive learning and electric

shock learning as an aversive learning for flies. For the training of sugar learning, they first let the flies have first type of odor (CS+) via vertical tube and then gave them a sucrose filter paper as a reward. After this, they gave second type of odor (CS-) with a water filter paper so that the flies can learn which scent is matched with appetitive food. Similarly, for the training of electric shock, they gave CS+ to flies with electric shock as a punishment and then gave CS- without electric shock in order to make them memorize which scent is not good to eat. After these training processes, as a tool of checking their responses, they counted how many flies touched the sucrose filter paper for appetitive learning results and how many flies avoided the tube with CS+ as aversive learning performance results. Based on this experimental setup, first they blocked the octopamine synapses of the flies and observed how the flies responded differently in both aversive and appetitive learning. The results was that the flies barely changed their responsiveness to the CS+ odor in an aversive learning but showed big changes in their responsiveness to the sugar filter in an appetitive learning. More specifically, they did not show response to the sugar filter without octopamine in appetitive learning while still showed similar avoidance in aversive learning. They also tested the effects of dopamine on olfactory responsiveness by blocking dopaminergic chemical synapses of the other group of flies. The result was the flies showed fewer avoidance responses to the CS+ odor in an aversive learning but still showed appetitive responses to the sucrose filter. This experimental results imply that octopamine plays an important role in appetitive learning while the dopamine affects an aversive learning of olfactory system of the drosophila.

Dacks et al. explored the morphology of dopaminergic neurons in ALs of Manduca sexta and studied the effects of DA both on the odor-evoked responsiveness of AL neurons and on the aversive behavior learning process,

which are supported by well-designed and reasonable experiments [9]. As results, the dopaminergic neurons, DA-ir/TH-ir neurons, are spread in overall glomeruli of the ALs in the form of arch-like shape, which turns out to be important to improve the odor-evoked responses of AL neurons and to decreases the postexcitatory inhibition phase after the excitation. Moreover, DA plays a significant role in the building of aversive olfactory memory on a feeding behavioral level.

Two Effects of Dopamine Previously, we simply concluded dopamine is related to the aversive learning; however, interestingly dopaminergic neurons can show different behaviors when it binds to different type of receptors. There are two types of dopaminergic receptor: D1-like and D2-like receptors. If the dopamine binds to the D1-like receptor, it makes essential effects on reward-based learning with showing EPSP; or if it binds to the D2-like receptor, it involves aversive learning as we discovered [4] [33]. Also, it is known that the dopaminergic path with D1-like receptors are essential for instrumental learning [51].

Chapter 3

Artificial Neural Networks

This chapter provides the basic knowledge of supervised and and reward-based neural learning models. This chapter also describes full derivation for the generalized logistic regression arbitrary-depth learning and reward-based Hyperbolic Hebbian plasticity, which are the basic of building the CALM algorithms.

3.1 Overview of Artificial Neural Networks

Artificial neural networks (ANNs) is sophisticated computational learning method which mimics biological neural system which briefly covered in previous section. It is also referred to as connectionist system, parallel distributed processing, or neural computing [15]. In ANNs, each component name of a neuron introduced in Chapter 2.1 has different computational name. In this dissertation, each computational component name of a biological neuron is defined as follows. First, a neuron is called a node in ANNs. The synaptic strength between a presynaptic and postsynaptic neuron is called neural weight or simply weight; and a neural path from a presynaptic and post synaptic neuron is called a node link or simple link; and each link represents its neural weight. Note that weight has numerical values so it can be also named as weight parameter. Presynaptic action potential is called input value and postsynaptic action potential is called actual output value. Activation level (potential) in biology is referred to as net value which is through the computational process

between input values and neural weights. Actual output value is given through the activation function or step function which takes net value as function input. Note that a net node represents the net value and an actual output node has activated value which is actual output value through the activation function; therefore each net node links to each corresponding actual output node. The detailed computational process of an ANN is different based on different learning types. We will see several different learning processes in the following subsections.

An activation function defines the way of generating action potential for an actual output node and each node can be through each different activation function if it is necessary. Briefly, there are three types of activation functions: (1) step function (e.g., binary function, bipolar function, etc.), (2) linear function, (3) non-linear function (e.g., logistic (sigmoid), hyperbolic tangent, Gaussian function, etc.). For an instance, if an output node takes the activation function as bipolar, it shows spike when the net value is $+1$, which can be regarded as EPSP in biology neural system, otherwise it will have -1 as IPSP.

Typically, input data into ANNs is given and weights are initially set with random values. The goal of ANNs is to find most appropriate weight parameter(s) based on given data and network topology. In other words, the learning process is about how to adjust the neural weights until the ultimate learning result is satisfied by a learning designer; therefore so-called 'learning rule' indicates 'how to update/adjust the weights of ANNs', which is the core of categorizing learning types. Different learning methods are depending on which kind of weight update learning rule is used; so the way of adjusting weight parameters decides ANN learning type such as supervised learning, unsupervised learning, reinforcement learning, or hybrid learning.

If not only input but also output are given to an ANN, the way of adjusting weights follows supervised learning. Supervised learning in an ANN is, briefly, the computing process by which the weights of the ANN are adjusted by decreasing the error between the actual output of the network and the desired output of given training data [11]. This means that, for supervised learning, we must have a training set for which we have a known desired output, target output, or label, for each input. The idea is that the trained ANN can then be used to label new data (for generalization). An arbitrary-depth neural networks refers to an ANN through at least three layers in computing the final actual output [5].

On the other hand, if only inputs are given to a network, the way of adjusting weights follows unsupervised learning. Unsupervised learning in ANNs is the process by which weights are learned by finding patterns or regularities between unlabeled input data. A typical application of unsupervised learning in ANNs is to group "similar" data points together into clusters, the number of which is typically small relative to the number of data points in the data set.

Unlike both supervised and unsupervised learning, reinforcement learning requires feedback (reward/punishment) as an evaluative signal from an environment, which says how appropriate the output was for given input. This signal allows a reinforcement learner to learn by trying various outputs and seeing which results in the greatest reward. Especially reinforcement learning has static state model and predict which sate should be next state based on current state by using a reward policy such as Sarsa, Q-learning, etc [64] [57]. Reward-based learning in an ANN is the computing process by applying the reward value directly to the weight update learning rule like modulatory Hebbian

learning. Note that reward-based neural learning is not based on the prediction model [38] [53].

It can not be said that which one is best among the various kinds of learning model. They have each different learning characteristics and different advantages on different type of domains therefore selecting a learning model depends on what kind of problem tasks are given. For example, a classification problem is better to be solved by supervised learning; and a problems of finding similarities among input fits to unsupervised learning such as k-means clustering. CALM is motivated from a supervised learning, reward-based learning, and several natural animal biological features, which aims to build a novel learning model towards a robot brain. CALM introduces how to take advantages of each learning method and shows how to overcome the limitations of each learning method with sound mathematical derivation and generalized synthetic experimental results.

3.2 Supervised Neural Learning Model

In order to build a novel learning model based on a existing model, it is important to understand the latter's principles in depth. An ANN can be classified according to a type of cost function and weight update rule. Cost function is defined for evaluating the learning status of an ANN (e.g., least mean squared (LMS) error or delta rule, logistic regression, etc.). Weight update rule is an optimization process based on the defined cost function (e.g., gradient descent (GD), conjugate gradient descent (CGD), etc.). In this dissertation, an arbitrary-depth neural learning with logistic regression backpropagation and gradient descent optimization is covered thoroughly. It is explained step by

step with three sub categories for incremental understanding of its profound computational process and principles: (1) A perceptron, (2) 2-layered, and (3) generalized, arbitrary-depth, ANN with logistic regression backpropagation (LRB). This section takes a bulk of this dissertation because CALM is based on the underlying principles of the arbitrary-depth neural learning with logistic regression.

3.2.1 A Perceptron ANN with Logistic Regression

In this section, we will see a simple ANN from the basics to its computational process in depth.

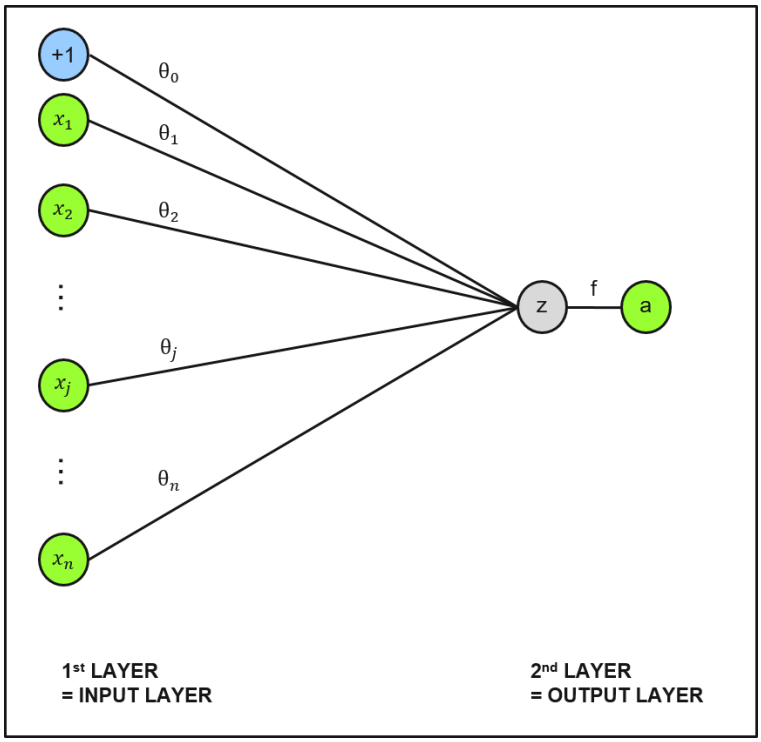


Figure 3.1: Perceptron ANN Architecture

A Perceptron ANN Architecture The Figure 3.1 shows a perceptron ANN architecture. A perceptron consists of nodes and links and each link has weight parameter representing the synaptic strength between two nodes. Nodes are classified into four types: (1) bias (+1), (2) input (x_j), (3) net (z), and (4) actual output node (a). Each weight is denoted as θ_j. As shown in Figure 3.1, a perceptron takes $n+1$ number of input values including bias value and gives out one actual output after activating one net node value. In this dissertation, we will call the $n+1$ number of input values as one input vector, input data example, or simply one input data. Similarly, we will call the $n+1$ number of weights as the weight vector, which is from all input nodes toward the net node. The goal of a perceptron ANN is binary classification. In other words, a perceptron aims to make itself give appropriate actual output value (0 or 1) for each input data example so that all given input data examples could be classified into either 0 or 1. The way of classifying each input data into either 0 or 1 is to adjust or update the weight vector based on the error between a given target output and processed actual output value. Thus it can be said that a perceptron is to optimize the weight vector based on given input vectors and target output values. The detail computational process is described in the learning paragraph.

A Perceptron ANN Data Table The Figure 3.2 shows a possible form of a perceptron ANN data table which can be used for understanding a perceptron ANN learning process. Also, this data table can be used for a developer to come up with how to log the ANN learning process or how to prepare a training data including input and target output. Also, the EKB (Experience-based Knowledge Base) is based on this data table, which is a component of CALM explained in Chapter 5.

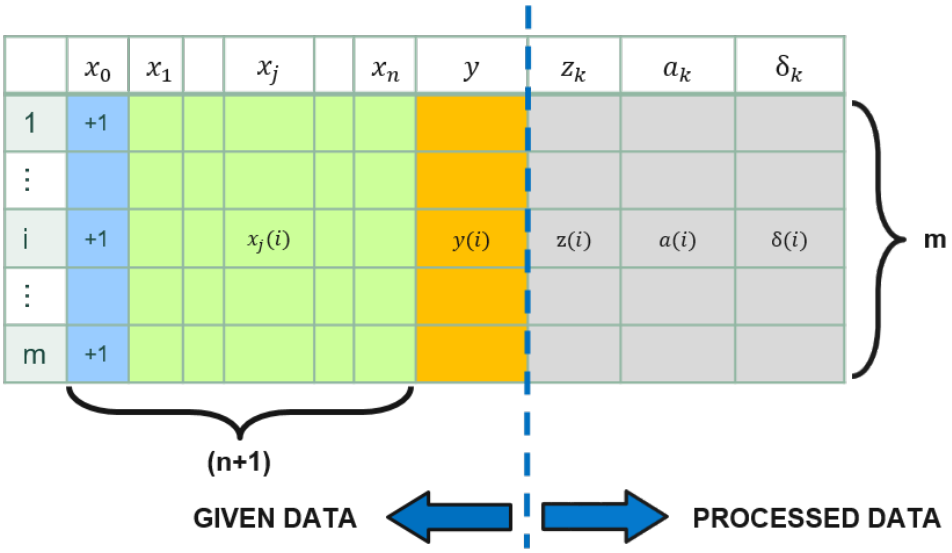


Figure 3.2: Perceptron ANN Data Table

For the notation in a perceptron ANN learning, m is the total number of given input data examples (input size), n is the number of given input features or attributes (input dimension), i indicates index of each input data example which can be up to m, and j is the index of each input feature (input node) which can be up to n. θ indicates the weight vector which represent the links from all input nodes toward the one net node. Based on this, $x_j(i)$ is j^{th} input node value on i^{th} input data, $y(i)$ is target output value on i^{th} input data, $z(i)$ is the net node value of i^{th} input data, $a(i)$ is the processed actual output value of i^{th} input data, $\delta(i)$ is the error value between the actual output value and target output value of i^{th} input data.

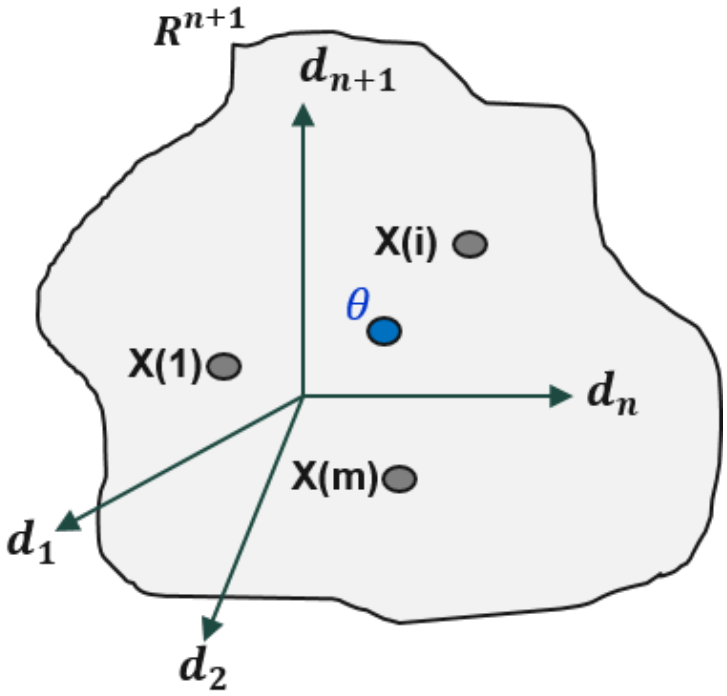


Figure 3.3: Perceptron ANN Data Space

The most interesting part of ANN computation is revealed when it is compared to general scientific computation. Figure 3.3 shows how a perceptron ANN can be interpreted in a different view. In $n+1$ dimensional space, each input vector can be represented as $X(i)$ and weight vector θ will be optimized over the learning process based on each input vector and target output value. This is important point of view in understanding the following learning process in depth.

Batch Learning - Vectorwise The Algorithm 1 shows pseudocode for a perceptron ANN with logistic regression and gradient descent optimization on vectorwise batch learning. Note that the term 'vectorwise' means the ANN compute the learning process by taking each input vector, $X(i)$, sequentially; on the other hand, 'matrixwise' means the ANN compute the learning process by taking all m number of input vectors at once. In this paragraph, we will first see The detailed process of vectorwise learning by looking into the Algorithm 1.

Algorithm 1 Perceptron ANN Batch Learning Vectorwise Pseudocode

Get $X \in R^{m \times n}$, $y \in R^{m \times 1}$, T, m, n, η, λ.
Add $\forall_i,\ X_0(i) = +1$ for bias $\rightarrow X \in R^{m \times (n+1)}$
Init $\theta \in R^{(n+1) \times 1}$
for $t = 1$ **to** T **do**
  for $i = 1$ **to** m **do**
    $z(i) = \theta^\top(t) X(i)$
    $a(i) = f(z(i))$
    $\delta(i) = a(i) - y(i)$
    $Cost(i) = -y(i) ln(a(i)) - (1 - y(i)) ln(1 - a(i))$
    $Cost(t) = Cost(t) + Cost(i)$
    $\Delta\theta(i) = \delta(i) X(i)$
    $\Delta\theta(t) = \Delta\theta(t) + \Delta\theta(i)$
  end for
  $R(t) = \frac{\lambda}{2m} \sum_{j=1}^{n} (\theta_j(t))^2$
  $J(t) = \frac{1}{m} Cost(t) + R(t)$
  $\theta(t+1) = \theta(t) - \eta \left(\frac{1}{m} \Delta\theta(t) + \frac{\lambda}{m} \bar{\theta}(t) \right)$
end for

• Input vector $X(i)$ represents each input data example which is each row of given data in the Figure 3.2. Note that the first element of a input vector is always 1 since it represents a bias node.

$$X(i) = \begin{bmatrix} x_0(i) \\ x_1(i) \\ \vdots \\ x_j(i) \\ \vdots \\ x_n(i) \end{bmatrix}_{(n+1) \times 1} \quad \text{where } x_0(i) = 1 \text{ for the bias.}$$

$$x_j(i) = j^{th} \text{ input feature value(node value) on } i^{th} \text{ example.}$$

• Weight vector θ is as follows. Note that there is another form of weight vector

$\bar{\theta}$ is defined as follows which will be used in computing regularization term. $\bar{\theta}$ is same as θ except for the first element.

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_j \\ \vdots \\ \theta_n \end{bmatrix}_{(n+1)\times 1} ; \ \bar{\theta} = \begin{bmatrix} 0 \\ \theta_1 \\ \vdots \\ \theta_j \\ \vdots \\ \theta_n \end{bmatrix}_{(n+1)\times 1} \quad \text{where } \theta_0 = 0$$

• Net value $z(i)$ is calculated by multiplying each element of input vector and weight vector as follows; and the computing process is named as net process in this dissertation.

$$\begin{aligned} z(i) &= \theta^\top X(i) \\ &= \theta_0 x_0(i) + \theta_1 x_1(i) + \cdots + \theta_n x_n(i) \\ &= \theta_0 + \theta_1 x_1(i) + \cdots + \theta_n x_n(i) \\ &= \sum_{j=0}^{n} \theta_j x_j(i) \in (-\infty, \infty) \end{aligned}$$

Note that the net process is inner product which is linear computation so it can be described as shown in Figure 3.4 on a general scientific computational view. It can be also said that a perceptron is same as linear regression model if it only considers net process.

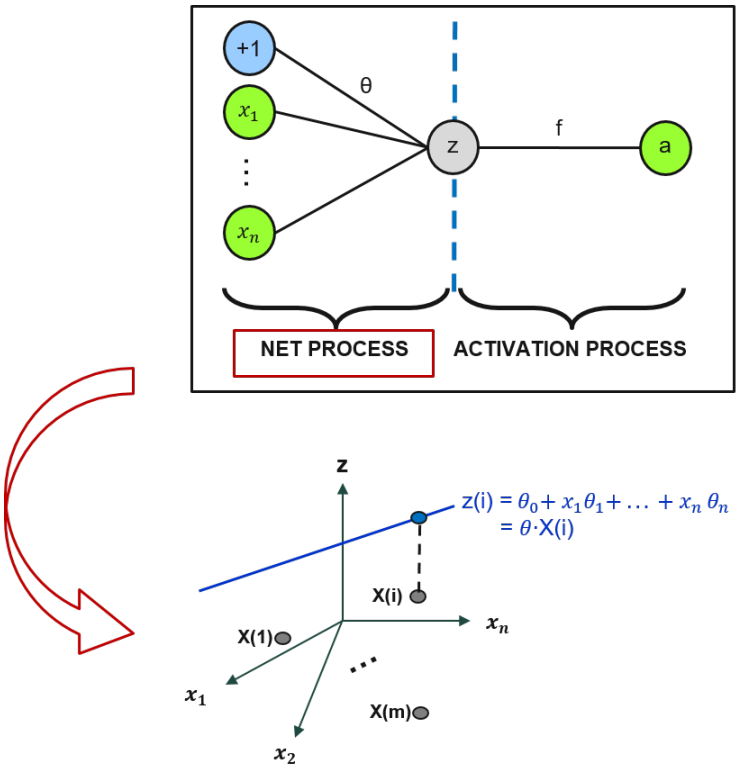


Figure 3.4: Perceptron ANN Net Process

• Actual output value $a(i)$ is an activated value of $a(i)$ and computed as follows. It is calculated by applying a certain type of activation function into the net node value, which is called activation process in this dissertation. In this logistic regression neural model, the activation function is logistic function which is also called sigmoid function. Figure 3.5 shows a graph of logistic regression activation function which explains the relationship between each net value and actual output value.

$$a(i) \quad = \quad f(z(i)) \in (0,1) \text{ where } f(z(i)) = \frac{1}{1+e^{z(i)}}$$

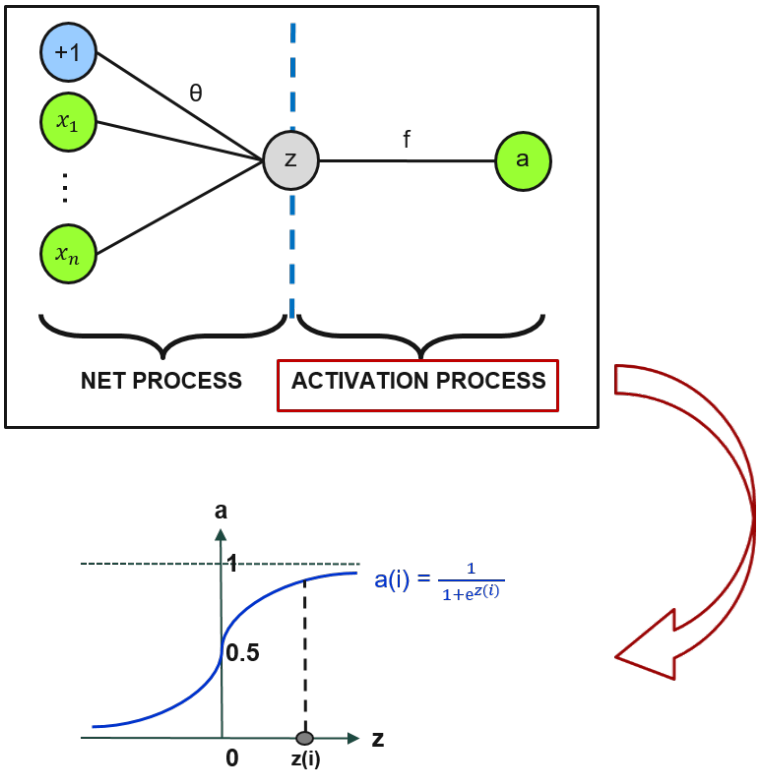


Figure 3.5: Perceptron ANN Activation Process

Note that if a net value is zero, the actual output value through the activation process is 0.5; and the possible output value of the activation function is between 0 and 1. This is very important feature of the logistic regression function since an actual output value can be fairly compared to the target output value which has either 0 or 1; if the range of $a(i)$ is out of between 0 and 1, it is hard to compare the meaning of actual output and target output. Also it is important that relationship between net node and actual output is monotonic increasing or decreasing; if $z(i)$ increases $a(i)$ also increases monotonically. Likewise, if $z(i)$ decreases, $a(i)$ also decreases.

• Target output $y(i)$ is a numerical value corresponding to a input vector $X(i)$, which is either 0 or 1. In a perceptron ANN, if a target output value is 1, this

means the corresponding input data $X(i)$ is classified into the target; or if the target output is 0, the input data is non-classified into a target, which represent binary classification. Note that this target output is also given as well as the input data as shown in the Figure 3.2.

$$y(i) \in \{0, 1\}$$

• Error value $\delta(i)$ represents the difference between the actual output value $a(i)$ and the target output value $y(i)$, which is between -1 and 1.

$$\delta(i) \quad = \quad a(i) - y(i) \in (-1, 1)$$

Note that there are three possible cases of an error value $\delta(i)$ since $y(i)$ has either 0 or 1 and the cases are organized as follows. First, if the error value is zero then it means the actual and target output values are exactly same which is the ultimate goal of ANN learning. Second, if the error is positive then it means the actual output is greater than target output value and thus the the goal of learning is to make the actual value decreased. Third, if the error value is negative then it means the actual output is smaller than the target output value and thus the goal of learning is to increase the actual output value.

$$\begin{cases} \delta(i) = 0 \iff a(i) = y(i) \\ \delta(i) > 0 \iff a(i) > y(i) \implies a(i) > 0 \ \& \ y(i) = 0 \\ \delta(i) < 0 \iff a(i) < y(i) \implies a(i) < 1 \ \& \ y(i) = 1 \end{cases}$$

• The cost function $J(\theta)$ gives a way of measuring learning results or learning effects based on errors between actual and target output values. The role of the cost function is to give smaller value when the actual and target output have similar value or gives larger value when the error is large so that a learner can recognize current learning status. In other words, cost function shows correlation between actual output and target output in current learning step and the ultimate goal of a learning is to have smaller cost value over learning steps. Equation (3.1) is the cost function defined for a perceptron ANN with logistic regression.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(-y(i)ln(a(i)) - (1-y(i))ln(1-a(i))) + \underbrace{\frac{\lambda}{2m}\sum_{j=1}^{n}(\theta_j)^2}_{\text{Regularization Term}} \quad (3.1)$$

$$= \frac{1}{m}\sum_{i=1}^{m} Cost(i) + R$$

$$\text{where } \begin{cases} Cost(i) = -y(i)ln(a(i)) - (1-y(i))ln(1-a(i)) \\ R = \frac{\lambda}{2m}\sum_{j=1}^{n}(\theta_j)^2 \end{cases}$$

Looking deep into Equation (3.1), $Cost(i)$ in the equation can be divided as two cases depending on a target output value as follows. When a target output $y(i)$ is 1, $Cost(i)$ gives out smaller value when the actual output $a(i)$ is closer to $y(i)$ which is 1. Otherwise, if the target output $y(i)$ is 0, $Cost(i)$ gives smaller value when $a(i)$ is closer to 0. In this way, $J(\theta)$ can be the indicator representing learning effects, which is the average of all $Cost(i)$ for each input vector and target output value. Figure 3.6 helps us to visually understand how a perceptron evaluates its cost depending on a given input and target output.

$$Cost(i) = -y(i)ln(a(i)) - (1 - y(i))ln(1 - a(i))$$

If $y(i) = 1$

$Cost(i) = -ln(a(i))$

$y(i) = 1, a(i) = 1 \rightarrow \delta(i) = 0 \rightarrow Cost(i) = 0$

$y(i) = 1, a(i) < 1 \rightarrow \delta(i) < 0 \rightarrow Cost(i) \uparrow$

If $y(i) = 0$

$Cost(i) = -ln(1 - a(i))$

$y(i) = 0, a(i) > 0 \rightarrow \delta(i) > 0 \rightarrow Cost(i) \uparrow$

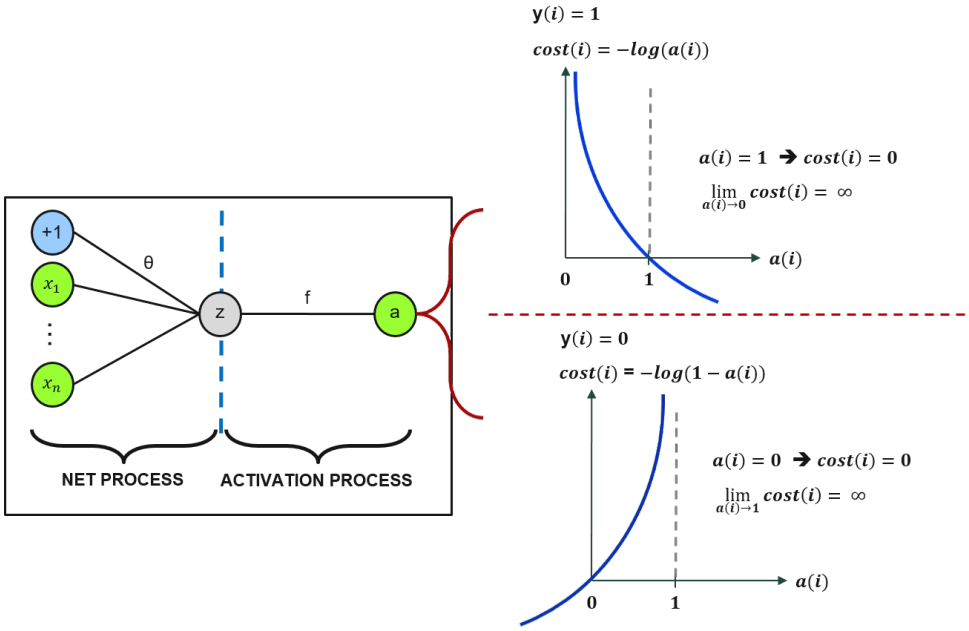


Figure 3.6: Perceptron ANN Role of Cost Function

• Weight Update - Elementwise. Based on the cost function $J(\theta)$ the weight update rule for a perceptron ANN is shown in Equation (3.2). The basic idea of the weight update rule is to find the local minimum, which is the lowest partial gradient value of the cost function over each weight parameter, and adjust each weight parameter toward the local minimum. This method is called gradient descent method.

$$\begin{aligned}
\theta_j(t+1) &= \theta_j(t) - \eta(\frac{\partial}{\partial\theta_j}J(\theta)) \\
&= \theta_j(t) - \eta(\Delta\theta_j) \qquad (3.2) \\
\text{where } \Delta\theta_j &= \frac{1}{m}\sum_{i=1}^{m}\delta(i)x_j(i) + \underbrace{\frac{\lambda}{m}\theta_{j\ (j\geq 1)}}_{\text{Regularization Term}} \\
&= \frac{1}{m}\sum_{i=1}^{m}(a(i)-y(i))x_j(i) + \frac{\lambda}{m}\theta_{j\ (j\geq 1)}
\end{aligned}$$

Note that 'update' is to adjust current weight through a certain type of weight update rule such as gradient descent, conjugate gradient descent, etc. The updated weight vector is denoted as $\theta(t+1)$ which refers to the adjusted weight vector $\theta(t)$ by the weight update rule at learning step t. This updated weight vector $\theta(t+1)$ will be used in calculating net process and activation process at next learning step $t+1$. Also, it is notable that there are three way of updating weight: (1) elementwise, (2) vectorwise, and (3) matrixwise weight update; 'elementwise' refers to update each weight parameter θ_j, 'vectorwise' is to update weight vector θ at once, and 'matrixwise' refers to update θ at once when θ has matrix form in 2-layered or generalized, arbitrary-depth, ANN.

• Weight Update - Vectorwise. Equation (3.2) can be written in a vectorwise

form as following Equation (3.3), which is self-explanatory.

$$\begin{aligned}\theta(t+1) &= \theta(t) - \eta(\frac{\Delta}{\Delta\theta}J(\theta)) \\ &= \theta(t) - \eta(\Delta\theta) \qquad (3.3)\\ \text{where } \Delta\theta &= \frac{1}{m}\sum_{i=1}^{m}\delta(i)X(i) + \underbrace{\frac{\lambda}{m}\bar{\theta}}_{\text{Regularization Term}}\end{aligned}$$

$$\Delta\theta = \begin{bmatrix}\Delta\theta_0\\ \Delta\theta_1\\ \vdots\\ \Delta\theta_j\\ \vdots\\ \Delta\theta_n\end{bmatrix} = \frac{1}{m}\sum_{i=1}^{m}\delta(i)\begin{bmatrix}x_0(i)\\ x_1(i)\\ \vdots\\ x_j(i)\\ \vdots\\ x_n(i)\end{bmatrix} + \frac{\lambda}{m}\bar{\theta}$$

Looking deep into the Equation (3.3), we can understand the role of the weight update rule, which is the gradient descent optimization. In order to understand how to adjust the weight vector, we will simplify the Equation (3.3) by assuming the learning parameters as follows.

$$\text{Assume: } m = 1, \lambda = 0, \text{ and } \eta = 1$$
$$\text{Then: } \theta(t+1) = \theta(t) - \delta(i)X(i)$$

In the simplified equation, weight update rule is vector sum based on the error value $\delta(i)$, which makes the weight vector θ closer to or farther away from an input vector $X(i)$. We know there are three possible cases of $\delta(i)$ and thus there are three cases of the weight vector is updated. We will see how the weight vector is adjusted in each case. First, if $\delta(i) = 0$, then $a(i)$ and $y(i)$ have same

value and thus the weight update rule is described as follows. In this case, the weight vector stays on current position after updating.

$$\begin{cases} \delta(i) = 0 \ (a(i) - y(i) = 0) & \implies \Delta\theta = 0 \\ & \implies \theta(t+1) = \theta(t) \end{cases}$$

Second, if $\delta(i) > 0$, this means $a(i) > y(i)$ and it implies that $y(i) = 0$ and $a(i)$ should be decreased in next learning step in order to have lower and lower value of cost function over learning steps. In order to decrease the value of $a(i)$, the weight update rule makes θ farther away from the current input vector, $X(i)$, so as to make the $z(i)$ value is decreased at next learning step. Consequently, $a(i)$ will be also decreased at next learning step since the relationship between $z(i)$ and $a(i)$ is monotonic. This process is well-organized as follows and Figure 3.7 helps us visually understand this process, which is the role of weight update when the error value is positive.

$$
\begin{cases}
\delta(i) > 0 \ (a(i) > y(i)) \\
\implies \Delta\theta = +\delta(i)X(i) \\
\implies \theta(t+1) = \theta(t) - \delta(i)X(i) \\
\implies \theta(t+1) \text{ will be farther away from } X(i) \text{ compared to } \theta(t) \\
\quad \text{(refer to the Figure 3.7, blue vector)} \\
\implies \theta(t+1) \cdot X(i) < \theta(t) \cdot X(i) \\
\implies z(i) \text{ will be decreased at next learning step} \\
\implies a(i) = f(z(i)) \text{ will be decreased at next learning step} \\
\implies \delta(i) \text{ will be decreased at next learning step} \\
\implies Cost(i) \text{ will be decreased at next learning step}
\end{cases}
$$

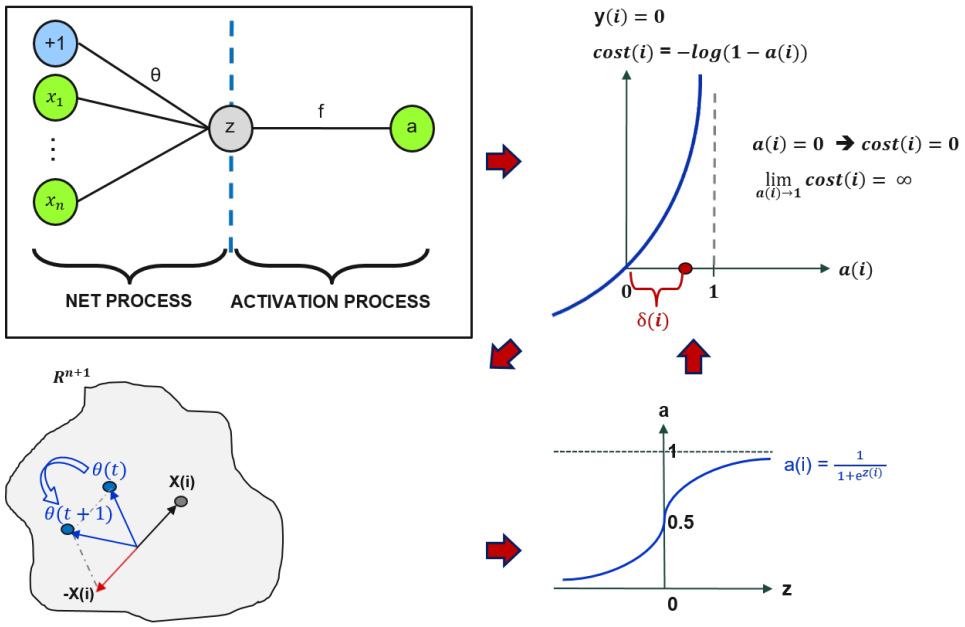


Figure 3.7: Perceptron ANN Role of Weight Update when $\delta(i) > 0$

Lastly, if $\delta(i) < 0$, this means $a(i) < y(i)$ and it implies that $y(i) = 1$ and $a(i)$ should be increased in next learning step in order to have lower and lower value of cost function over learning steps. In order to increase the value of $a(i)$, the weight update rule makes θ closer to the current input vector, $X(i)$, so as to make the $z(i)$ value is increased at next learning step. Consequently, $a(i)$ will be also increased at next learning step since the relationship between $z(i)$ and $a(i)$ is monotonic. This process is well-organized as follows and Figure 3.8 helps us visually understand this process, which is the role of weight update when the error value is negative.

$$
\begin{cases}
\delta(i) < 0 \ (a(i) < y(i)) \\
\implies \Delta\theta = -\delta X(i) \\
\implies \theta(t+1) = \theta(t) + \delta X(i) \\
\implies \theta(t+1) \text{ will be closer to} X(i) \text{ compared to } \theta(t) \\
\qquad \text{(refer to the Figure 3.8, blue vector)} \\
\implies \theta(t+1) \cdot X(i) > \theta(t) \cdot X(i) \\
\implies z(i) \text{ will be increased at next learning step} \\
\implies a(i) = f(z(i)) \text{ will be increased at next learning step} \\
\implies \delta(i) \text{ will be decreased at next learning step} \\
\implies Cost(i) \text{ will be decreased at next learning step}
\end{cases}
$$

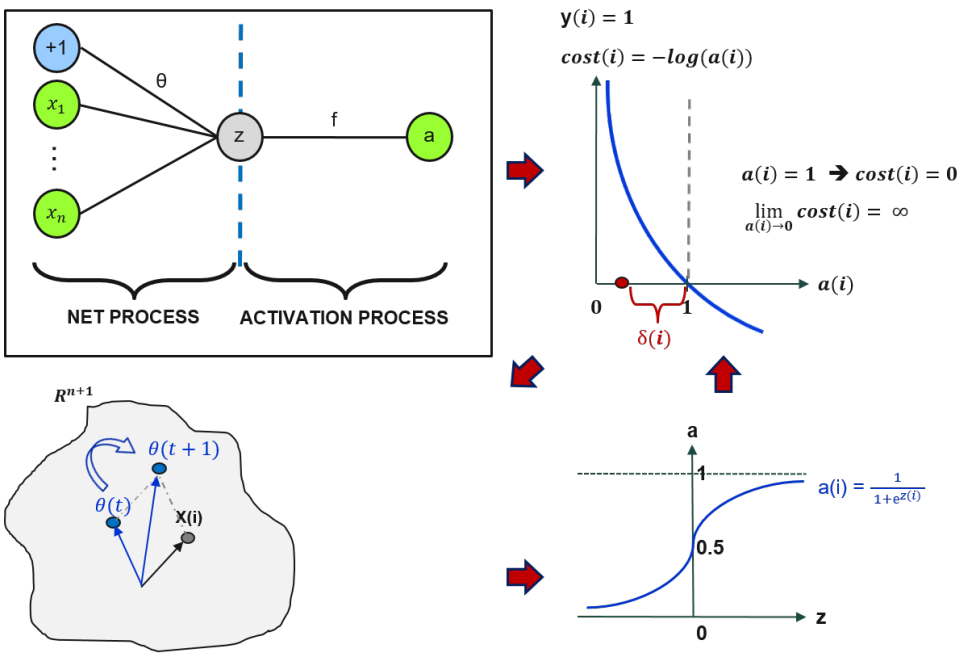


Figure 3.8: Perceptron ANN Role of Weight Update when $\delta(i) < 0$

In this way, the weight vector θ is updated over learning steps and it can be considered as a binary classifier. Over the learning process, θ decides which kind of input vectors, $X(i)$s, should keep closer or keep away based on the cost function and the weight update rule.

Batch Learning - Matrixwise The Algorithm 2 shows pseudocode for a perceptron ANN with logistic regression and gradient descent optimization on matrixwise batch learning. In this paragraph, we will see how a perceptron ANN takes the whole input and target output at once, so-called matrixwise batch learning. Note that it is based on vectorwise batch learning and the learning results of both are same, but the matrixwise computational speed would be faster than vectorwise because there are less for loop.

Algorithm 2 Perceptron ANN Batch Learning Matrixwise Pseudocode

Get $X \in R^{m \times n}$, $y \in R^{m \times 1}$, T, m, n, η, λ.
Add $\forall_i, X_0(i) = +1$ for bias $\rightarrow X \in R^{m \times (n+1)}$
Init $\theta \in R^{(n+1) \times 1}$
for $t = 1$ **to** T **do**
  $z = X\theta(t)$
  $a = f(z)$
  $\delta = a - y$
  $Cost(t) = \frac{1}{m} \sum_{i=1}^{m} (-y(i)ln(a(i)) - (1 - y(i))ln(1 - a(i)))$
  $\Delta\theta(t) = X^\top \delta$
  $R(t) = \frac{\lambda}{2m} \sum_{j=1}^{n} (\theta_j)^2$
  $J(t) = Cost(t) + R(t)$
  $\theta(t+1) = \theta(t) - \eta \left(\frac{1}{m} \Delta\theta(t) + \frac{\lambda}{m} \bar{\theta}(t) \right)$
end for

- Input X is matrix covering all m number of input vectors as follows.

$$X = \begin{bmatrix} X(1)^\top \longrightarrow \\ X(2)^\top \longrightarrow \\ \vdots \\ X(i)^\top \longrightarrow \\ \vdots \\ X(m)^\top \longrightarrow \end{bmatrix}_{m \times (n+1)} \quad \text{where } X(i) = \begin{bmatrix} x_0(i) \\ x_1(i) \\ \vdots \\ x_j(i) \\ \vdots \\ x_n(i) \end{bmatrix}_{(n+1) \times 1}$$

• Weight vector θ is same as in vectorwise batch learning.

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_j \\ \vdots \\ \theta_n \end{bmatrix}_{(n+1)\times 1} ; \ \bar{\theta} = \begin{bmatrix} 0 \\ \theta_1 \\ \vdots \\ \theta_j \\ \vdots \\ \theta_n \end{bmatrix}_{(n+1)\times 1} \quad \text{where } \theta_0 = 0$$

• Net vector z represents all values of net process as follows.

$$z = \begin{bmatrix} z(1) \\ z(2) \\ \vdots \\ z(i) \\ \vdots \\ z(m) \end{bmatrix}_{m\times 1} = \begin{bmatrix} \theta^\top X(1) \\ \theta^\top X(2) \\ \vdots \\ \theta^\top X(i) \\ \vdots \\ \theta^\top X(m) \end{bmatrix}_{m\times 1} = \underbrace{X\theta}_{\text{linear system}}$$

$$\begin{aligned} \text{where } z(i) &= \theta_0 x_0(i) + \theta_1 x_1(i) + \cdots + \theta_n x_n(i) \\ &= \sum_{j=0}^{n} \theta_j x_j(i) \in (-\infty, \infty) \end{aligned}$$

• Actual output vector a represents all m number fo actual output values as

follows. f is the same logistic function as in vectorwise batch learning.

$$a = \begin{bmatrix} a(1) \\ a(2) \\ \vdots \\ a(i) \\ \vdots \\ a(m) \end{bmatrix}_{m \times 1} = f(z)$$

• Target output vector y represents all m number of target output values as follows.

$$y = \begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(i) \\ \vdots \\ y(m) \end{bmatrix}_{m \times 1}$$

• Error vector δ represents all m number of differences between actual and target

output values as follows.

$$\delta \;=\; \begin{bmatrix} \delta(1) \\ \delta(2) \\ \vdots \\ \delta(i) \\ \vdots \\ \delta(m) \end{bmatrix}_{m\times 1} = a - y$$

• The cost function $J(\theta)$ also has exactly same meaning in vectorwise batch learning but the computational process is changed as follows since it handles changed mathematical forms.

$$J(\theta) == \frac{1}{m}\sum_{i=1}^{m}(-y(i)ln(a(i)) - (1-y(i))ln(1-a(i))) + \frac{\lambda}{2m}\sum_{j=1}^{n}(\theta_j)^2 \quad (3.4)$$

• Weight Update in matrixwise batch learning is also same as vectorwise learning but the computational process is changed as follows.

$$\theta(t+1) = \theta(t) - \eta(\frac{1}{m}(X^\top\delta) + \frac{\lambda}{m}\bar{\theta}) \quad (3.5)$$
$$(= \theta(t) - \eta(\frac{1}{m}\sum_{i=1}^{m}\delta(i)X(i) + \frac{\lambda}{m}\bar{\theta}))$$

Derivation of Weight Update Rule This paragraph derives the Equation (3.2). The mathematical process for the derivation fully described each step by step to be easily understood as follows.

$$
\begin{aligned}
\theta_j(t+1) &= \theta_j(t) - \eta(\tfrac{\partial}{\partial\theta_j}J(\theta)) \\
&= \theta_j(t) - \eta(\Delta\theta_j) \\
\Delta\theta_j &= \tfrac{\partial}{\partial\theta_j}J(\theta) \\
&= \tfrac{\partial}{\partial\theta_j}\left(\tfrac{1}{m}\sum_{i=1}^{m}(-y(i)ln(a(i)) - (1-y(i))ln(1-a(i)))\right) \\
&\quad + \tfrac{\partial}{\partial\theta_j}\left(\tfrac{\lambda}{2m}\sum_{j=1}^{n}(\theta_j)^2\right) \\
&= \tfrac{\partial}{\partial\theta_j}\left(\tfrac{1}{m}\sum_{i=0}^{m}Cost(i) + R\right)
\end{aligned}
$$

$$
\begin{aligned}
\tfrac{\partial}{\partial\theta_j}Cost(i) &= \tfrac{\partial}{\partial\theta_j}(-y(i)ln(a(i)) - (1-y(i))ln(1-a(i))) \\
&= \tfrac{\partial}{\partial a(i)}Cost(i)\cdot\tfrac{\partial a(i)}{\partial z(i)}\cdot\tfrac{\partial z(i)}{\partial\theta_j} \\
\tfrac{\partial}{\partial a(i)}Cost(i) &= \tfrac{-y(i)}{a(i)} - \tfrac{1-y(i)}{1-a(i)}\cdot(1-a(i))'(\because ln(x)' = \tfrac{1}{x}) \\
&= \tfrac{-y(i)}{a(i)} + \tfrac{1-y(i)}{1-a(i)} \\
&= \tfrac{a(i)-a(i)\cdot y(i)-y(i)+a(i)\cdot y(i)}{a(i)(1-a(i))} \\
&= \tfrac{a(i)-y(i)}{a(i)(1-a(i))} \\
&= \tfrac{\delta(i)}{a(i)(1-a(i))}
\end{aligned}
$$

$$
\begin{aligned}
\tfrac{\partial a(i)}{\partial z(i)} &= \tfrac{\partial f(z(i))}{\partial z(i)} \\
&= f(z(i))(1-f(z(i)))(\because f(x) = \tfrac{1}{1+e^{-x}},\ f'(x) = f(x)(1-f(x))) \\
&= a(i)(1-a(i)) \\
\tfrac{\partial z(i)}{\partial\theta_j} &= \tfrac{\partial}{\partial\theta_j}(\sum_{j=1}^{n}\theta_j x_j(i)) \\
&= x_j(i)
\end{aligned}
$$

$$
\begin{aligned}
\therefore \frac{\partial}{\partial \theta_j} Cost(i) &= \frac{\delta(i)}{a(i)(1-a(i))} \cdot \frac{a(i)(1-a(i))}{1} \cdot x_j(i) \\
&= \delta(i) \cdot x_j(i) \\
\frac{\partial}{\partial \theta_j} R &= \frac{\partial}{\partial \theta_j} (\frac{\lambda}{2m} \sum_{j=1}^{n} (\theta_j)^2) \\
&= \frac{\lambda}{m} \theta_j \ _{(j \geq 1)} \\
\therefore \Delta \theta_j &= \frac{1}{m} \sum_{i=1}^{m} \delta(i) \cdot x_j(i) + \frac{\lambda}{m} \theta_j \ _{(j \geq 1)} \\
\therefore \theta_j(t+1) &= \theta_j(t) - \eta(\frac{1}{m} \sum_{i=1}^{m} \delta(i) \cdot x_j(i) + \frac{\lambda}{m} \theta_j \ _{(j \geq 1)})
\end{aligned}
$$

3.2.2 2-Layered ANN with Logistic Regression

In this section, we will see from the basic of a 2-layered ANN to its computational process in depth, which is based on a perceptron ANN.

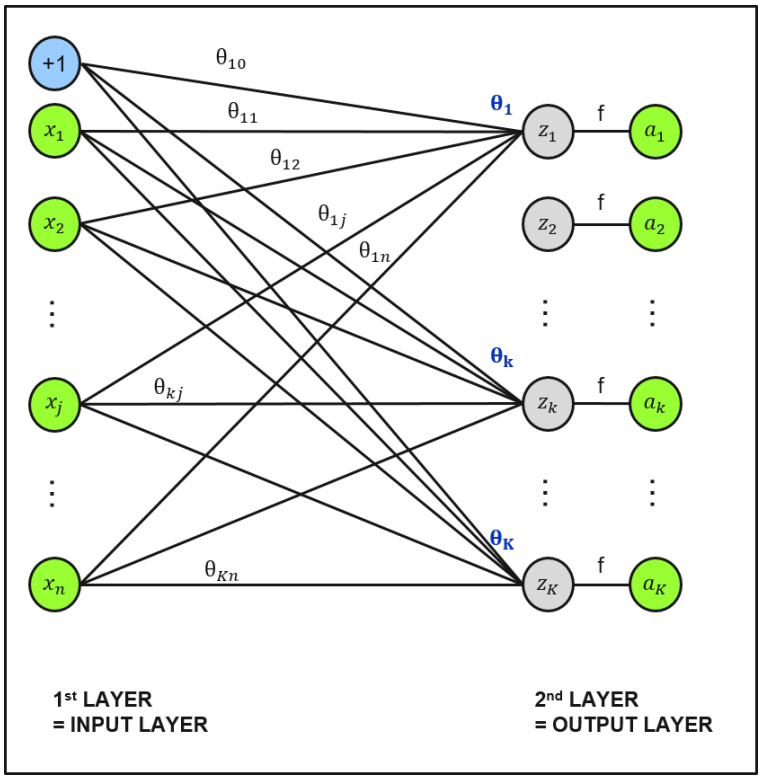


Figure 3.9: 2-Layered ANN Architecture

2-Layered ANN Architecture The Figure 3.9 shows a 2-layered ANN architecture, which has multiple output nodes. Similar to a perceptron ANN, nodes are classified into four types: (1) bias (+1), (2) input (x_j), (3) net (z_k), and (4) actual output node (a_k). Each weight is denoted as θ_{kj} which represent strength of the link from input node x_j to net node z_k. The big difference from a perceptron is there are K number of weight vectors each of which is denoted as θ_k covering weight parameters from all input nodes toward each net node z_k. For example, θ_3 is the weight vector covering weight parameters from all inputs toward the net node z_3. As shown in Figure 3.9, a 2-layered ANN takes $n+1$ number of input values including bias and gives out K number of actual output values through each activation process. The goal of a 2-layered ANN is multi-class classification, more specifically K-class classification. It aims to make

itself select appropriate actual output node for given each input example so that all given input data examples can be classified one of K actual output nodes. Note that, given a input data example, the selecting one among K number of actual output nodes is processed by finding maximum actual output value. The way of classifying each input data into one of K classes is to adjust or update K number of weight vectors based on errors between given target and processed actual output values. The details is described in the learning paragraph.

2-Layered ANN Data Table Compared to the Figure 3.2, the data table for 2-layered ANN has additional columns for covering multiple outputs as shown in the Figure 3.10.

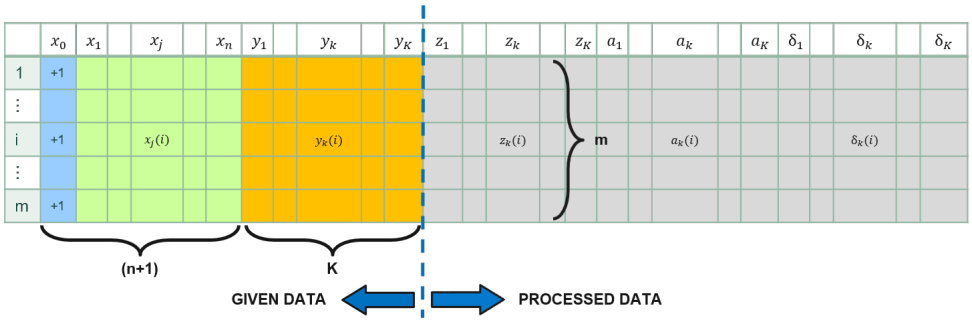


Figure 3.10: 2-Layered ANN Data Table

For the notation in 2-layered ANN learning, m is the total number of given input data examples (input size), n is the number of given input features or attributes (input dimension), i indicates index of each input data example which can be up to m, and j is the index of each input feature (input node) which can be up to n. Upper case K refers to the number of actual output nodes and lower case k is the indicator of each actual output node which can be up to K. θ_k indicates the weight vector which represent the links from all input nodes toward the net node z_k. Based on this, $x_j(i)$ is j^{th} input node value on i^{th} input

data, $y_k(i)$ is target output value corresponding to k^{th} actual output node on i^{th} input data, $z_k(i)$ is the k^{th} net node value of i^{th} input data, $a_k(i)$ is the k^{th} processed actual output node value of i^{th} input data, $\delta_k(i)$ is the k^{th} error value between the actual output node $a_k(i)$ and target output value $y_k(i)$ of i^{th} input data.

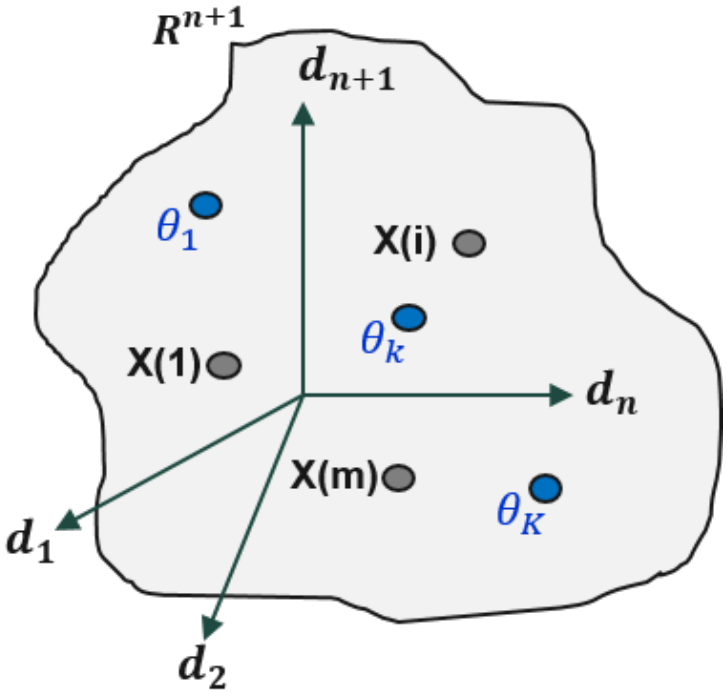


Figure 3.11: 2-Layered ANN Data Space

Likewise, it is interesting when a 2-layered ANN is compared to general scientific computation. Figure 3.11 shows how a 2-layered ANN can be described in a different view. In $n+1$ dimensional space, each input vector can be represented as $X(i)$ and each weight vector θ_k will be optimized over learning process based on each input $X(i)$ and target output value $y_k(i)$.

Batch Learning - Vectorwise The Algorithm 3 shows pseudocode for a 2-layered ANN with logistic regression and gradient descent optimization on

vectorwise batch learning.

Algorithm 3 2-Layered ANN Batch Learning Vectorwise Pseudocode

Get $X \in R^{m \times n}$, $y \in R^{m \times K}$, $T, m, n, K, \eta, \lambda$.
Add $\forall_i, X_0(i) = +1$ for bias $\rightarrow X \in R^{m \times (n+1)}$
Init $\theta \in R^{K \times (n+1)}$
for $t = 1$ **to** T **do**
  for $i = 1$ **to** m **do**
    $z(i) = \theta(t)X(i)$
    $a(i) = f(z(i))$
    $\delta(i) = a(i) - y(i)$
    $Cost(i) = \sum_{k=1}^{K}(-y_k(i)ln(a_k(i)) - (1 - y_k(i))ln(1 - a_k(i)))$
    $Cost(t) = Cost(t) + Cost(i)$
    $\Delta\theta(i) = \delta(i)X(i)^\top$
    $\Delta\theta(t) = \Delta\theta(t) + \Delta\theta(i)$
  end for
  $R(t) = \frac{\lambda}{2m} \sum_{k=1}^{K} \sum_{j=1}^{n} (\theta_{kj}(t))^2$
  $J(t) = \frac{1}{m} Cost(t) + R(t)$
  $\theta(t+1) = \theta(t) - \eta \left(\frac{1}{m} \Delta\theta(t) + \frac{\lambda}{m} \bar{\theta}(t) \right)$
end for

- Input vector $X(i)$ represents each input data.

$$X(i) = \begin{bmatrix} x_0(i) \\ x_1(i) \\ \vdots \\ x_j(i) \\ \vdots \\ x_n(i) \end{bmatrix}_{(n+1) \times 1} \quad \text{where } x_0(i) = 1 \text{ for the bias.}$$

- Weight matrix θ and weight vector θ_k are as follows. Note that θ refers to all

neural links in the ANN and θ_k indicates each weight vector of it.

$$\theta = \begin{bmatrix} \theta_1^\top \longrightarrow \\ \theta_2^\top \longrightarrow \\ \vdots \\ \theta_k^\top \longrightarrow \\ \vdots \\ \theta_K^\top \longrightarrow \end{bmatrix}_{K\times(n+1)} ; \ \theta_k = \begin{bmatrix} \theta_{k0} \\ \theta_{k1} \\ \vdots \\ \theta_{kj} \\ \vdots \\ \theta_{kn} \end{bmatrix}_{(n+1)\times 1} ; \ \bar{\theta_k} = \begin{bmatrix} 0 \\ \theta_{k1} \\ \vdots \\ \theta_{kj} \\ \vdots \\ \theta_{kn} \end{bmatrix} \text{ where } \theta_{k0} = 0$$

• Net vector $z(i)$ represents all K number of net values of i^{th} input data and computed as follows.

$$\begin{aligned} z(i) &= \begin{bmatrix} z_1(i) \\ z_2(i) \\ \vdots \\ z_k(i) \\ \vdots \\ z_K(i) \end{bmatrix}_{K\times 1} = \theta X(i) \\ z_k(i) &= \theta_k^\top X(i) \\ &= \theta_{k0}x_0(i) + \theta_{k1}x_1(i) + \cdots + \theta_{kn}x_n(i) \\ &= \sum_{j=0}^{n} \theta_{kj}x_j(i) \in (-\infty, \infty) \end{aligned}$$

Note that each net process $z_k(i)$ is inner product which is linear computation so it can be said that there are K number of linear processes as shown in

Figure 3.4 on a general scientific computational view. Therefore, it is interesting that a 2-layered ANN can be considered as multivariate linear regression model if it only has K number of net processes without activation processes.

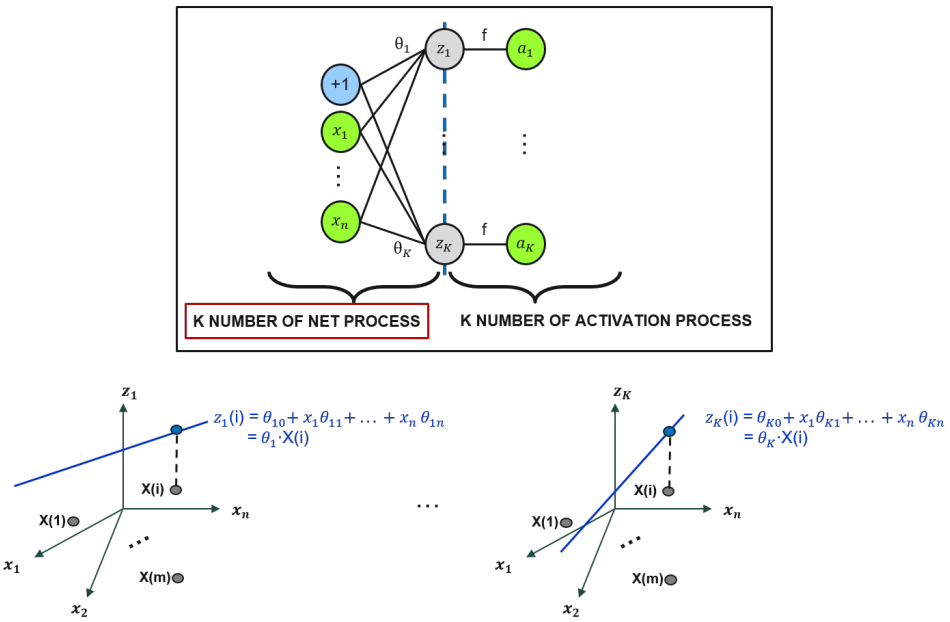


Figure 3.12: 2-Layered ANN Net Process

• Actual output vector $a(i)$ represents all K number of actual output values of i^{th} input data and it is computed as follows. Similar to a perceptron, Figure 3.13 shows that there are K number of activation processes each of which is associated

with corresponding net process.

$$a(i) = \begin{bmatrix} a_1(i) \\ a_2(i) \\ \vdots \\ a_k(i) \\ \vdots \\ a_K(i) \end{bmatrix}_{K\times 1} = f(z(i)) \text{ where } f(z_k(i)) = \frac{1}{1+e^{z_k(i)}}$$

$$a_k(i) = f(z_k(i)) \in (0,1)$$

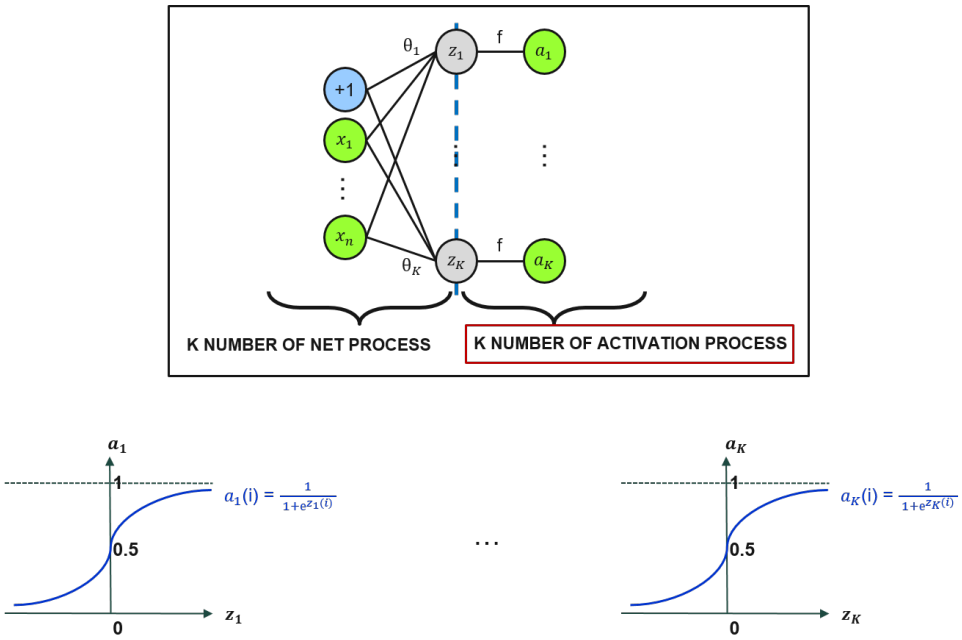


Figure 3.13: 2-Layered ANN Activation Process

• Target output vector $y(i)$ represents K number of target output values of i^{th} input data and it is computed as follows. Note that each target output value

has either 0 or 1.

$$\begin{aligned} y(i) &= \begin{bmatrix} y_1(i) \\ y_2(i) \\ \vdots \\ y_k(i) \\ \vdots \\ y_K(i) \end{bmatrix}_{K\times 1} \\ y_k(i) &\in \{0,\, 1\} \end{aligned}$$

• Error vector $\delta(i)$ represents K number of error values and it is computed as follows.

$$\begin{aligned} \delta(i) &= \begin{bmatrix} \delta_1(i) \\ \delta_2(i) \\ \vdots \\ \delta_k(i) \\ \vdots \\ \delta_K(i) \end{bmatrix}_{K\times 1} = a(i) - y(i) \\ \delta_k(i) &= a_k(i) - y_k(i) \in (-1, 1) \end{aligned}$$

Likewise, there three possible cases of an each error value $\delta_k(i)$ since $y_k(i)$ has either 0 or 1 as follows.

$$\begin{cases} \delta_k(i) = 0 \iff a_k(i) = y_k(i) \\ \delta_k(i) > 0 \iff a_k(i) > y_k(i) \implies a_k(i) > 0 \;\&\; y_k(i) = 0 \\ \delta_k(i) < 0 \iff a_k(i) < y_k(i) \implies a_k(i) < 1 \;\&\; y_k(i) = 1 \end{cases}$$

• The cost function $J(\theta)$ in a 2-layered ANN also has same role as one in a perceptron; the difference is that it considers K number of different errors for a given one input data and thus the function definition is changed to Equation (3.6) from Equation (3.1).

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}(-y_k(i)ln(a_k(i)) - (1-y_k(i))ln(1-a_k(i))) + \underbrace{\frac{\lambda}{2m}\sum_{k=1}^{K}\sum_{j=1}^{n}(\theta_{kj})^2}_{\text{Regularization Term}} \tag{3.6}$$

$$= \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} Cost_k(i) + R$$

$$\text{where } \begin{cases} Cost_k(i) = (-y_k(i)ln(a_k(i)) - (1-y_k(i))ln(1-a_k(i))) \\ R = \frac{\lambda}{2m}(\sum_{k=1}^{K}\sum_{j=1}^{n}(\theta_{kj})^2) \end{cases}$$

Looking deep into Equation (3.6), $Cost_k(i)$ in the equation can be divided as two cases as follows. When a target output $y_k(i)$ is 1, $Cost_k(i)$ gives smaller value when the actual output $a_k(i)$ is closer to $y_k(i)$. Otherwise, if the target output $y(i)_k$ is 0, $Cost_k(i)$ gives smaller value when $a_k(i)$ is closer to 0. In this way, $J(\theta)$ can be the indicator representing learning effects, which is the average of all $Cost_k(i)$ for all input vectors. Figure 3.14 helps us to visually understand how a 2-layered ANN evaluates its cost depending on a given input and target output.

$$Cost_k(i) \quad = \quad -y_k(i)ln(a_k(i)) - (1-y_k(i))ln(1-a_k(i))$$

If $y_k(i) = 1$

$Cost_k(i) = -ln(a_k(i))$

$y_k(i) = 1, a_k(i) = 1 \rightarrow \delta_k(i) = 0 \rightarrow Cost_k(i) = 0$

$y_k(i) = 1, a_k(i) < 1 \rightarrow \delta_k(i) < 0 \rightarrow Cost_k(i) \uparrow$

If $y_k(i) = 0$

$Cost_k(i) = -ln(1 - a_k(i))$

$y_k(i) = 0, a_k(i) = 0 \rightarrow \delta_k(i) = 0 \rightarrow Cost_k(i) = 0$

$y_k(i) = 0, a_k(i) > 0 \rightarrow \delta_k(i) > 0 \rightarrow Cost_k(i) \uparrow$

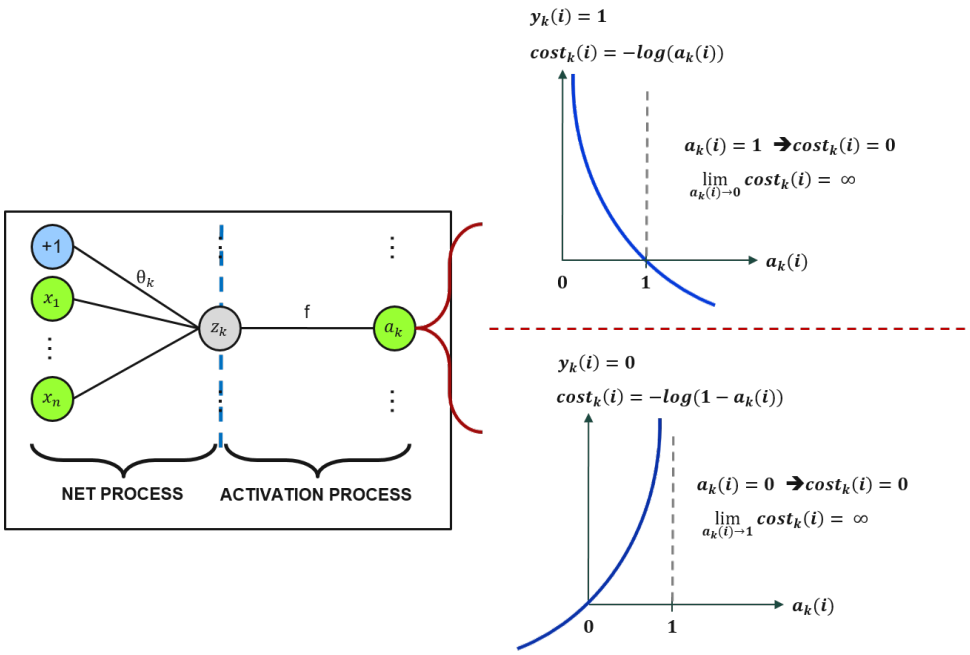


Figure 3.14: 2-Layered ANN Role of Cost Function

Note that having K number of $Cost_k(i)$ is the core principle of the K-class classification. This implies the important point that if there are K number of output nodes, there are K number of corresponding $Cost_k(i)$ which checks the

difference between each actual and target output value. And cost function, $J(\theta)$, tells us the overall average value of how different between all actual and target output through the total number of given input data examples.

• Weight Update - Elementwise. Based on the cost function $J(\theta)$ the weight update rule is shown in Equation (3.7).

$$\begin{aligned}\theta_{kj}(t+1) &= \theta_{kj}(t) - \eta(\frac{\partial}{\partial\theta_{kj}}J(\theta)) \\ &= \theta_{kj}(t) - \eta(\Delta\theta_{kj}) \\ \text{where } \Delta\theta_{kj} &= \frac{1}{m}\sum_{i=1}^{m}\delta_k(i)x_j(i) + \underbrace{\frac{\lambda}{m}\theta_{kj\ (j\geq 1)}}_{\text{Regularization Term}} \\ &= \frac{1}{m}\sum_{i=1}^{m}(a_k(i) - y_k(i))x_j(i) + \frac{\lambda}{m}\theta_{kj\ (j\geq 1)}\end{aligned} \tag{3.7}$$

• Weight Update - Vectorwise. Equation (3.7) can be written in a vectorwise form as following Equation (3.8), which is self-explanatory.

$$\begin{aligned}\theta_k(t+1) &= \theta(t) - \eta(\frac{\partial}{\partial\theta_k}J(\theta)) \\ &= \theta_k(t) - \eta(\Delta\theta_k) \\ \text{where } \Delta\theta_k &= \frac{1}{m}\sum_{i=1}^{m}\delta_k(i)X(i) + \frac{\lambda}{m}\bar{\theta_k} \\ \Delta\theta_k &= \begin{bmatrix}\Delta\theta_{k0}\\ \Delta\theta_{k1}\\ \vdots\\ \Delta\theta_{kj}\\ \vdots\\ \Delta\theta_{kn}\end{bmatrix}_{(n+1)\times 1} = \frac{1}{m}\sum_{i=1}^{m}\delta_k(i)\begin{bmatrix}x_0(i)\\ x_1(i)\\ \vdots\\ x_j(i)\\ \vdots\\ x_n(i)\end{bmatrix}_{(n+1)\times 1} + \frac{\lambda}{m}\bar{\theta_k}\end{aligned} \tag{3.8}$$

Looking deep into the Equation (3.8), we can understand the role of the weight update rule in a 2-layered ANN. Likewise, we will simplify the Equation (3.8) by assuming the learning parameters as follows.

$$\text{Assume: } m = 1, \lambda = 0, \text{ and } \eta = 1$$
$$\text{Then: } \theta_k(t+1) = \theta_k(t) - \delta_k(i)X(i)$$

In this case, we know there are three possible cases of $\delta_k(i)$ and thus there are three cases of each weight vector θ_k is updated; We will see how each weight vector is adjusted by the weight update rule in three cases. First, if $\delta_k(i) = 0$, this means $a_k(i)$ and $y_k(i)$ have same value and thus the weight update rule is described as follows. In this case, the weight vector θ_k stay on current position in next learning step.

$$\begin{cases} \delta_k(i) = 0 \ (a_k(i) - y_k(i) = 0) & \implies \Delta\theta_k = 0 \\ & \implies \theta_k(t+1) = \theta_k(t) \end{cases}$$

Second, if $\delta_k(i) > 0$, this means $a_k(i) > y_k(i)$ and it implies that $y_k(i) = 0$ and $a_k(i)$ should be decreased in next learning step in order to have lower and lower value of cost function over learning steps. In order to decrease the value of $a_k(i)$, the weight update rule makes θ_k farther away from the current input vector, $X(i)$, so as to make the $z(i)$ value is decreased at next learning step. Consequently, $a_k(i)$ will be also decreased at next learning step since the relationship between $z_k(i)$ and $a_k(i)$ is monotonic. This process is well-described as follows and Figure 3.15 helps us visually understand this process, which is

the role of weight update when the error value is positive.

$$
\begin{cases}
\delta_k(i) > 0 \ (a_k(i) > y_k(i)) \\
\implies \Delta\theta_k = +\delta_k(i)X(i) \\
\implies \theta_k(t+1) = \theta_k(t) - \delta_k(i)X(i) \\
\implies \theta_k(t+1) \text{ will be farther away from } X(i) \text{ compared to } \theta_k(t) \\
\quad \text{(refer to the Figure 3.7, blue vector)} \\
\implies \theta_k(t+1) \cdot X(i) < \theta_k(t) \cdot X(i) \\
\implies z_k(i) \text{ will be decreased at next learning step} \\
\implies a_k(i) = f(z_k(i)) \text{ will be decreased at next learning step} \\
\implies \delta_k(i) \text{ will be decreased at next learning step} \\
\implies Cost_k(i) \text{ will be decreased at next learning step}
\end{cases}
$$

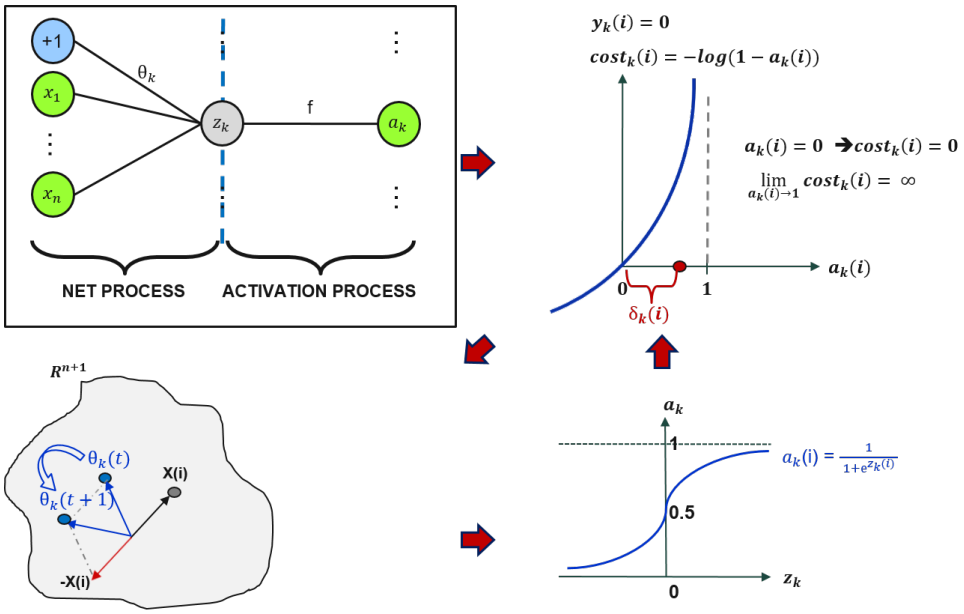


Figure 3.15: 2-Layered ANN Role of Weight Update when $\delta_k(i) > 0$

Lastly, if $\delta_k(i) < 0$, this means $a_k(i) < y_k(i)$ and it implies that $y_k(i) = 1$ and $a_k(i)$ should be increased in next learning step in order to have lower and lower value of cost function over learning steps. In order to increase the value of $a_k(i)$, the weight update rule makes θ_k closer to the current input vector, $X(i)$, so as to make the $z_k(i)$ value is increased at next learning step. Consequently, $a_k(i)$ will be also increased at next learning step since the relationship between $z_k(i)$ and $a_k(i)$ is monotonic. This process is well-described as follows and Figure 3.8 helps us visually understand this process, which is the role of weight update when the error value is negative.

$$
\begin{cases}
\delta_k(i) < 0 \ (a(i) < y(i)) \\
\implies \Delta\theta_k = -\delta X(i) \\
\implies \theta_k(t+1) = \theta_k(t) + \delta_k X(i) \\
\implies \theta_k(t+1) \text{ will be closer to} X(i) \text{ compared to } \theta_k(t) \\
\qquad \text{(refer to the Figure 3.8, blue vector)} \\
\implies \theta_k(t+1) \cdot X(i) > \theta_k(t) \cdot X(i) \\
\implies z_k(i) \text{ will be increased at next learning step} \\
\implies a_k(i) = f(z_k(i)) \text{ will be increased at next learning step} \\
\implies \delta_k(i) \text{ will be decreased at next learning step} \\
\implies Cost_k(i) \text{ will be decreased at next learning step}
\end{cases}
$$

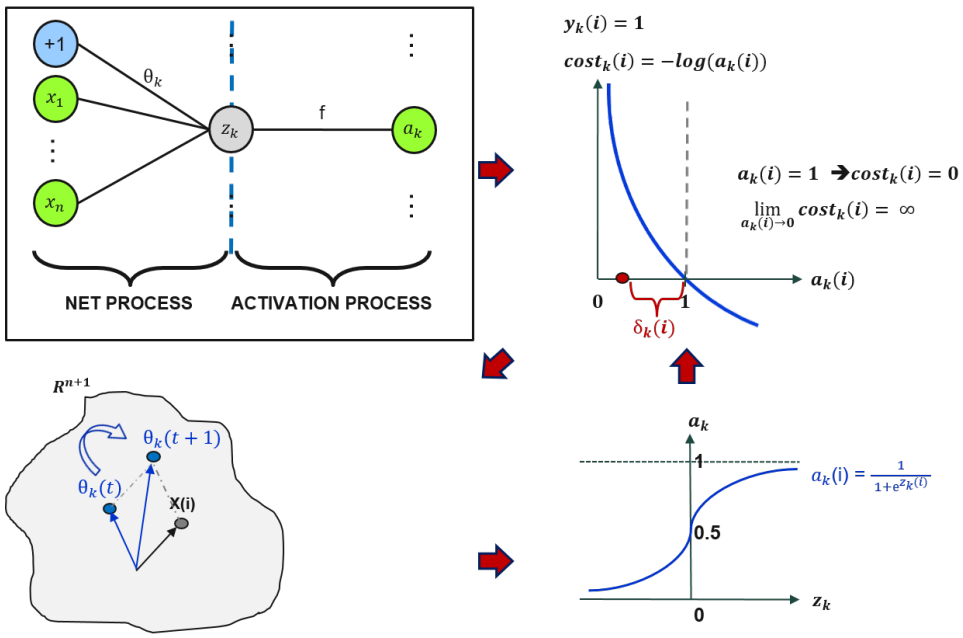


Figure 3.16: 2-Layered ANN Role of Weight Update when $\delta_k(i) < 0$

Note that if there are K number of target output, there are K number of classifiers, denoted as θ_k, and each of which takes charge in classifying each input data based on each error value, $\delta_k(i)$. In other words, each θ_k decides which kind of input vectors, $X(i)$s, should keep closer or keep away from itself through the weight update rule. Thus, if input space has more than or equal K number of cluster groups, then having less than K number of classifiers in 2-layered ANN is not enough to classify all input. So, the number of Input clusters, C_X, should be less than or equals to K in 2-layered ANN. This limitations is the reason why arbitrary-depth ANN needs to solve more complex input space, which is discussed in Chapter 6, 7, and 9.

• Weight Update - matrixwise. Compared to a perceptron ANN, 2-layered has weight matrix θ which covers all weight vectors in the ANN. The matrixwise

weight update rule is shown Equation (3.9), which is based on Equation (3.8).

$$\begin{aligned}\theta(t+1) &= \theta(t) - \eta(\frac{\Delta}{\Delta\theta}J(\theta)) \\ &= \theta(t) - \eta(\Delta\theta) \qquad (3.9)\\ \text{where } \Delta\theta &= \frac{1}{m}\sum_{i=1}^{m}\delta(i)X(i)^\top + \frac{\lambda}{m}\bar{\theta}\end{aligned}$$

$$\Delta\theta = \begin{bmatrix} \Delta\theta_1^\top \longrightarrow \\ \Delta\theta_2^\top \longrightarrow \\ \vdots \\ \Delta\theta_k^\top \longrightarrow \\ \vdots \\ \Delta\theta_K^\top \longrightarrow \end{bmatrix}_{K\times(n+1)} = \frac{1}{m}\sum_{i=1}^{m}\delta(i)X(i)^\top + \frac{\lambda}{m}\bar{\theta}$$

Batch Learning - Matrixwise The Algorithm 4 shows pseudocode for a 2-layered ANN with logistic regression and gradient descent optimization on matrixwise batch learning. In this paragraph, we will see how it takes the whole input and target output at once, so-called matrixwise batch learning. Likewise, it is based on vectorwise batch learning and the learning results for both are same, but the matrixwise computational speed would be faster than vectorwise because there are less for loop.

Algorithm 4 2-Layered ANN Batch Learning Matrixwise Pseudocode

Get $X \in R^{m \times n}$, $y \in R^{m \times K}$, $T, m, n, K, \eta, \lambda$.
Add $\forall_i, X_0(i) = +1$ for bias $\rightarrow X \in R^{m \times (n+1)}$
Init $\theta \in R^{K \times (n+1)}$
for $t = 1$ **to** T **do**
  $z = X\theta^\top(t)$
  $a = f(z)$
  $\delta = a - y$
  $Cost(t) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} (-y_k(i) ln(a_k(i)) - (1 - y_k(i)) ln(1 - a_k(i)))$
  $\Delta\theta(t) = \delta^\top X$
  $R(t) = \frac{\lambda}{2m} \sum_{k=1}^{K} \sum_{j=1}^{n} (\theta_{kj}(t))^2$
  $J(t) = Cost(t) + R(t)$
  $\theta(t+1) = \theta(t) - \eta \left(\frac{1}{m} \Delta\theta(t) + \frac{\lambda}{m} \bar{\theta}(t) \right)$
end for

• Input X is matrix covering all m number of input vectors as follows, which is same as in vectorwise batch learning.

$$X = \begin{bmatrix} x(1)^\top \longrightarrow \\ x(2)^\top \longrightarrow \\ \vdots \\ x(i)^\top \longrightarrow \\ \vdots \\ x(m)^\top \longrightarrow \end{bmatrix}_{m \times (n+1)} ; \; X(i) = \begin{bmatrix} x_0(i) \\ x_1(i) \\ \vdots \\ x_j(i) \\ \vdots \\ x_n(i) \end{bmatrix}_{(n+1) \times 1}$$

• Weight matrix θ and weight vector θ_k as follows, which is same as in vectorwise

batch learning.

$$\theta = \begin{bmatrix} \theta_1^\top \longrightarrow \\ \theta_2^\top \longrightarrow \\ \vdots \\ \theta_k^\top \longrightarrow \\ \vdots \\ \theta_K^\top \longrightarrow \end{bmatrix}_{K\times(n+1)} ; \ \theta_k = \begin{bmatrix} \theta_{k0} \\ \theta_{k1} \\ \vdots \\ \theta_{kj} \\ \vdots \\ \theta_{kn} \end{bmatrix}_{(n+1)\times 1} ; \ \bar{\theta_k} = \begin{bmatrix} 0 \\ \theta_{k1} \\ \vdots \\ \theta_{kj} \\ \vdots \\ \theta_{kn} \end{bmatrix} \text{ where } \theta_{k0} = 0$$

- Net matrix z represents all m number of net vectors as follows.

$$z = \begin{bmatrix} z(1)^\top \longrightarrow \\ z(2)^\top \longrightarrow \\ \vdots \\ z(i)^\top \longrightarrow \\ \vdots \\ z(m)^\top \longrightarrow \end{bmatrix}_{m\times K} = \begin{bmatrix} X(1)\theta^\top \\ X(2)\theta^\top \\ \vdots \\ X(i)\theta^\top \\ \vdots \\ X(m)\theta^\top \end{bmatrix}_{m\times K} = X\theta^\top$$

$$z(i) = \begin{bmatrix} z_1(i) \\ z_2(i) \\ \vdots \\ z_k(i) \\ \vdots \\ z_K(i) \end{bmatrix}_{K\times 1} = \theta X(i)$$

- Actual output matrix a represents all m number of actual output vectors as

follows. f is the same logistic function as in vectorwise batch learning.

$$a = \begin{bmatrix} a(1)^\top \longrightarrow \\ a(2)^\top \longrightarrow \\ \vdots \\ a(i)^\top \longrightarrow \\ \vdots \\ a(m)^\top \longrightarrow \end{bmatrix}_{m \times K} = f(z)$$

• Target output matrix y represents all m number of target output vectors as follows.

$$y = \begin{bmatrix} y(1)^\top \longrightarrow \\ y(2)^\top \longrightarrow \\ \vdots \\ y(i)^\top \longrightarrow \\ \vdots \\ y(m)^\top \longrightarrow \end{bmatrix}_{m \times K}$$

• Error matrix δ represents all m number of error vectors as follows.

$$\delta = \begin{bmatrix} \delta(1)^\top \longrightarrow \\ \delta(2)^\top \longrightarrow \\ \vdots \\ \delta(i)^\top \longrightarrow \\ \vdots \\ \delta(m)^\top \longrightarrow \end{bmatrix}_{m \times K} = a - y$$

• The cost function for a 2-layered ANN on matrixwise batch learning is defined as follows. The results of this is same as the one in vectorwise learning but the calculation is different since it handles changed mathematical forms.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} (-y_k(i) ln(a_k(i)) - (1 - y_k(i)) ln(1 - a_k(i))) + \frac{\lambda}{2m} \sum_{k=1}^{K} \sum_{j=1}^{n} (\theta_{kj})^2 \tag{3.10}$$

• Weight Update for matrixwise batch learning is as follows.

$$\theta(t+1) = \theta(t) - \eta \left(\frac{1}{m} (\delta^\top X(i)) + \frac{\lambda}{m} \bar{\theta} \right) \left(= \theta(t) - \eta (\frac{1}{m} \sum_{i=1}^{m} (\delta(i) X(i)^\top) + \frac{\lambda}{m} \bar{\theta}) \right) \tag{3.11}$$

Derivation of Learning Update Rule This paragraph derives the Equation (3.7). The mathematical process for the derivation fully described each step by step to be easily understood as follows.

$$
\begin{aligned}
\theta_{kj}(t+1) &= \theta_{kj}(t) - \eta(\tfrac{\partial}{\partial\theta_{kj}}J(\theta)) \\
&= \theta_{kj}(t) - \eta(\Delta\theta_{kj}) \\
\Delta\theta_{kj} &= \tfrac{\partial}{\partial\theta_{kj}}J(\theta) \\
&= \tfrac{\partial}{\partial\theta_{kj}}\left(\tfrac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}(-y_k(i)ln(a_k(i)) - (1-y_k(i))ln(1-a_k(i)))\right) \\
&\quad + \tfrac{\partial}{\partial\theta_{kj}}\left(\tfrac{\lambda}{2m}\sum_{k=1}^{K}\sum_{j=1}^{n}(\theta_{kj})^2\right) \\
&= \tfrac{\partial}{\partial\theta_{kj}}\left(\tfrac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}Cost_k(i) + R\right)
\end{aligned}
$$

$$
\begin{aligned}
\tfrac{\partial}{\partial\theta_{kj}}(\sum_{k=1}^{K}Cost_k(i)) &= \tfrac{\partial}{\partial\theta_{kj}}Cost_k(i) \\
\tfrac{\partial}{\partial\theta_{kj}}Cost_k(i) &= \tfrac{\partial}{\partial\theta_{kj}}(-y_k(i)ln(a_k(i)) - (1-y_k(i))ln(1-a_k(i))) \\
&= \tfrac{\partial}{\partial a_k(i)}Cost_k(i)\cdot\tfrac{\partial a_k(i)}{\partial z_k(i)}\cdot\tfrac{\partial z_k(i)}{\partial\theta_{kj}} \\
\tfrac{\partial}{\partial a_k(i)}Cost_k(i) &= \tfrac{-y_k(i)}{a_k(i)} - \tfrac{1-y_k(i)}{1-a_k(i)}\cdot(1-a_k(i))'(\because ln(x)' = \tfrac{1}{x}) \\
&= \tfrac{-y_k(i)}{a_k(i)} + \tfrac{1-y_k(i)}{1-a_k(i)} \\
&= \tfrac{a_k(i)-a_k(i)\cdot y_k(i)-y_k(i)+a_k(i)\cdot y_k(i)}{a_k(i)(1-a_k(i))} \\
&= \tfrac{a_k(i)-y_k(i)}{a_k(i)(1-a_k(i))} \\
&= \tfrac{\delta_k(i)}{a_k(i)(1-a_k(i))}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial a_k(i)}{\partial z_k(i)} &= \frac{\partial f(z_k(i))}{\partial z_k(i)} \\
&= f(z_k(i))(1 - f(z_k(i)))(\because f(x) = \tfrac{1}{1+e^{-x}}, f'(x) = f(x)(1 - f(x))) \\
&= a_k(i)(1 - a_k(i)) \\
\frac{\partial z_k(i)}{\partial \theta_{kj}} &= \frac{\partial}{\partial \theta_{kj}} \sum_{j=1}^{n} \theta_{kj} x_j(i) \\
&= x_j(i) \\
\therefore \frac{\partial}{\partial \theta_{kj}} Cost_k(i) &= \frac{\delta_k(i)}{a_k(i)(1-a_k(i))} \cdot \frac{a_k(i)(1-a_k(i))}{1} \cdot x_j(i) \\
&= \delta_k(i) \cdot x_j(i) \\
\frac{\partial}{\partial \theta_{kj}} R &= \frac{\partial}{\partial \theta_{kj}} (\frac{\lambda}{2m} \sum_{k=1}^{K} \sum_{j=1}^{n} (\theta_{kj})^2) \\
&= \frac{\lambda}{m} \theta_{kj} \ _{(j \geq 1)} \\
\therefore \Delta \theta_{kj} &= \frac{1}{m} \sum_{i=1}^{m} \delta_k(i) \cdot x_j(i) + \frac{\lambda}{m} \theta_{kj} \ _{(j \geq 1)} \\
\therefore \theta_{kj}(t+1) &= \theta_{kj}(t) - \eta(\frac{1}{m} \sum_{i=1}^{m} \delta_k(i) \cdot x_j(i) + \frac{\lambda}{m} \theta_{kj} \ _{(j \geq 1)})
\end{aligned}
$$

3.2.3 Generalized Arbitrary-Depth ANN with Logistic Regression

In this section, we will see from the basic of a generalized, arbitrary-depth, ANN to its computational process in depth, which is based on a 2-layered ANN.

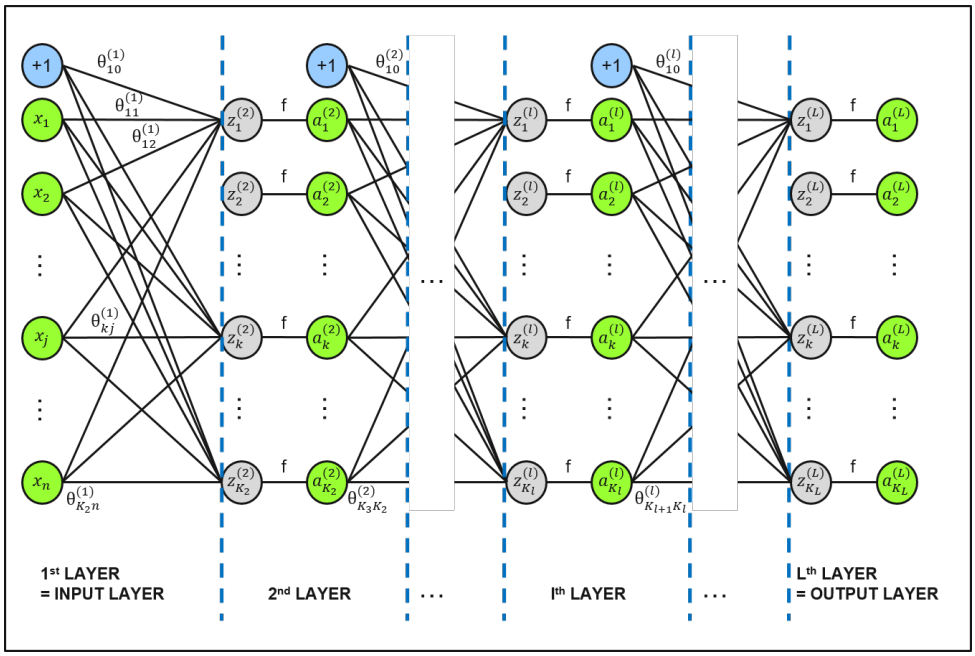


Figure 3.17: Generalized Arbitrary-Depth ANN Architecture

Generalized Arbitrary-Depth ANN Architecture The Figure 3.17 shows a generalized, arbitrary-depth, ANN architecture, which has multiple layers and output nodes. Unlike a 2-layered ANN, a generalized, arbitrary-depth, ANN has multiple layers where input layer is the first layer and the output layer is the last layer.

As shown in Figure 3.17, nodes are classified into four types: (1) bias(+1), (2) input(x_j), (3) net($z_k^{(l)}$), and (4) actual output node($a_k^{(l)}$). Each weight in each layer is denoted as $\theta_{kj}^{(l)}$ which represent strength of the link from an actual output node $a_j^{(l)}$ at current layer to a net node $z_k^{(l+1)}$ at next layer. $\theta_k^{(l)}$ represents a weight vector from all actual output nodes $\forall_j a_j^{(l)}$ at current layer to the net node $z_k^{(l+1)}$ at next layer.

Similar to 2-layered ANN, the goal of a generalized, arbitrary-depth, ANN is multi-class classification, specifically K_L-class classification but it is to solve

more complex input space where the 2-layered ANN has limitations to solve it, which is called arbitrary-depth learning. It aims to make itself select appropriate actual output node for given each input data example so that all given input data examples can be classified into one of K_L possible actual output nodes. The sound mathematical background for an arbitrary-depth neural learning with the logistic regression backpropagation is fully described in learning paragraph.

Generalized Arbitrary-Depth ANN Data Table Compared to the Figure 3.10, the data table for a generalized, arbitrary-depth, ANN has additional columns for covering multiple layers as shown in the Figure 3.18.

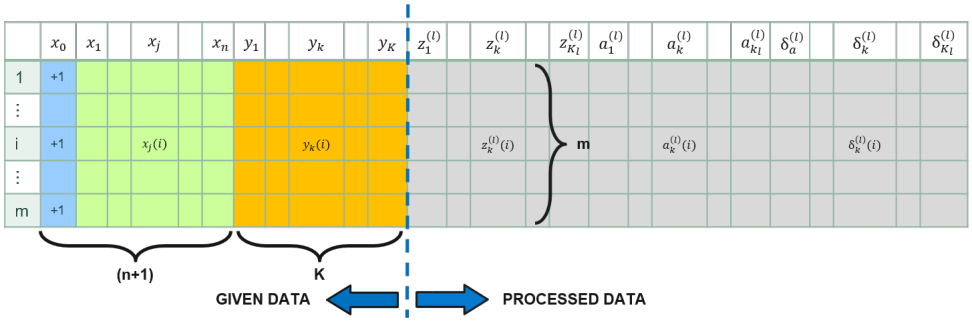


Figure 3.18: Generalized Arbitrary-Depth ANN Data Table

For the notation in a generalized, arbitrary-depth, ANN learning, m, n, i, j are same as in a 2-layered ANN learning. L is the number of total layers in an ANN including input and output layers, l refers to each layer which is denoted as superscript (l) where its value is from 1 to L, K_L is the number of actual output nodes at the last layer, K_l is the number of actual output nodes in each l^{th} layer. In here, it is notable that in a generalized, arbitrary-depth, ANN actual output nodes at each layer turns into the input nodes for the next layer. Therefore, for keeping consistency, n can be also denoted as same as K_1 since input layer is the first layer and thus x_j can be referred as a^1k where $k \in [0, K_1]$. Based on this,

$x_j(i)$ is j^{th} input node value on i^{th} input data which can be same as $a_k^{(1)}(i)$, $y_k(i)$ is target output value for $a_k^{(L)}$ of i^{th} input data, $z_k^{(l)}(i)$ is the k^{th} net node value at l^{th} layer of i^{th} input data, $a_k^{(l)}(i)$ is the the k^{th} processed actual output node value at l^{th} layer of i^{th} input data, $\delta_k^{(l)}(i)$ is the the k^{th} error value at l^{th} layer of i^{th} input data; especially, $\delta_k^{(L)}(i)$ is the the k^{th} error value between the actual output node value $a_k^{(L)}(i)$ and target output value $y_k(i)$. the computational process calculating $\delta_k^{(l)}(i)$ is covered in the next learning paragraph.

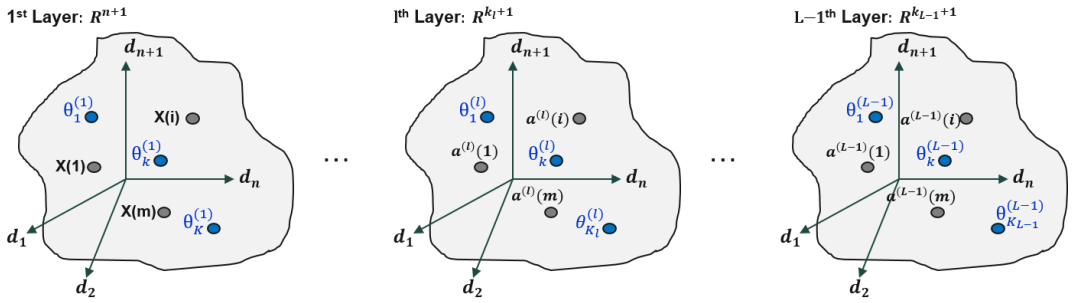


Figure 3.19: Generalized Arbitrary-Depth ANN Data Space

Likewise, it is interesting when a generalized, arbitrary-depth, ANN is compared to general scientific computation. Figure 3.19 shows how a generalized arbitrary-depth neural network can be described in a different view. Note that there are $L-1$ number of computing spaces each of which represents each layer; note that last L^{th} layer only has processed net and actual output values, without neural weights. At first layer, there is $n+1$ dimensional space where each input vector can be represented as $X(i)$ and each weight vector $\theta_k^{(1)}$ will be optimized over the learning process based on error value $\delta_k^{(1+1)}$. Similarly, l^{th} layer can be represented as K_l+1 dimensional space where each input is denoted as $a^{(l)}(i)$ and each weight vector $\theta_k^{(l)}$ is optimized over the learning process based on error value $\delta_k^{(l+1)}$. The detail optimization process is described in the following learning paragraph.

Batch Learning - Vectorwise The Algorithm 3 shows pseudocode for a generalized, arbitrary-depth, ANN with logistic regression and gradient descent optimization on vectorwise batch learning.

Algorithm 5 Generalized Arbitrary-depth ANN Batch Learning Vectorwise Pseudocode

Get $X \in R^{m \times n}$, $y \in R^{m \times K}$, $T, m, n, L, K_1, \cdots, K_l, \cdots, K_L, \eta, \lambda$,
Add $\forall_i, X_0(i) = +1$ for bias $\rightarrow X \in R^{m \times (n+1)}$
for $l = 1$ **to** $L - 1$ **do**
  Init $\theta^{(l)} \in R^{K_{l+1} \times (K_l + 1)}$
end for
for $t = 1$ **to** T **do**
  for $i = 1$ **to** m **do**
    FORWARD PROPAGATION
    $a^{(1)}(i) = X(i)$
    for l = 2 **to** L **do**
      $z^{(l)}(i) = \theta^{(l-1)}(t) a^{(l-1)}(i)$
      $a^{(l)}(i) = f(z^{(l)}(i))$
      Add $a_0^{(l)}(i) = +1$ for bias at each layer.
    end for
    BACKWARD PROPAGATION
    Remove $a_0^{(L)}(i)$ since there is no bias at the last layer.
    $\delta^{(L)}(i) = a^{(L)}(i) - y(i)$
    $\Delta\theta^{(L-1)}(i) = \delta^{(L)}(i)(a^{(L-1)}(i))^\top$
    $\Delta\theta^{(L-1)}(t) = \Delta\theta^{(L-1)}(t) + \Delta\theta^{(L-1)}(i)$
    for $l = L - 1$ **to** 2 **do**
      $\delta^{(l)}(i) = (\theta^{(l)})^\top \delta^{(l+1)}(i)$
      Remove $\delta_0^{(l)}$
      $\delta^{(l)}(i) = \delta^{(l)}(i) .* f(z^{(l)}(i))$
      $\Delta\theta^{(l-1)}(i) = \delta^{(l)}(i)(a^{(l-1)(i)})^\top$
      $\Delta\theta^{(l-1)}(t) = \Delta\theta^{(l-1)}(t) + \Delta\theta^{(l-1)}(i)$
    end for
    $Cost(i) = \sum_{k=1}^{K_L} (-y_k(i) ln(a_k^{(L)}(i)) - (1 - y_k(i)) ln(1 - a_k^{(L)}(i)))$
    $Cost(t) = Cost(t) + Cost(i)$
  end for
  WEIGHT UPDATE
  for l = 1 **to** L-1 **do**
    $\theta^{(l)}(t+1) = \theta^{(l)}(t) - \eta \left(\frac{1}{m} \Delta\theta^{(l)}(t) + \frac{\lambda}{m} \bar{\theta}^{(l)}(t) \right)$
  end for
  $R(t) = \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{k=1}^{K_{l+1}} \sum_{j=1}^{K_l} \left(\theta_{kj}^{(l)}(t) \right)^2$
  $J(t) = \frac{1}{m} Cost(t) + R(t)$
end for

• Input vector $X(i)$ represents each input vector.

$$X(i) = \begin{bmatrix} x_0(i) \\ x_1(i) \\ \vdots \\ x_j(i) \\ \vdots \\ x_n(i) \end{bmatrix}_{(n+1)\times 1} = a^{(1)}(i) = \begin{bmatrix} a_0^{(1)}(i) \\ a_1^{(1)}(i) \\ \vdots \\ a_j^{(1)}(i) \\ \vdots \\ a_{K_1}^{(1)}(i) \end{bmatrix}_{(K_1+1)\times 1}$$

where $x_0(i) = a_0(i) = 1$ and $K_1 = n$

• Weight matrix $\theta^{(l)}$ and weight vector $\theta_k^{(l)}$ at each layer are as follows. Note that l is from 1 to $L-1$ since there is no neural links at the last layer. $\theta^{(l)}$ refers to all weights at l^{th} layer in the ANN and $\theta_k^{(l)}$ indicates each weight vector of it.

$$\theta^{(l)} = \begin{bmatrix} \theta_1^{(l)\top} \longrightarrow \\ \theta_2^{(l)\top} \longrightarrow \\ \vdots \\ \theta_k^{(l)\top} \longrightarrow \\ \vdots \\ \theta_{K_{l+1}}^{(l)\top} \longrightarrow \end{bmatrix}_{K_{l+1}\times(K_l+1)} ; \ \theta_k^{(l)} = \begin{bmatrix} \theta_{k0}^{(l)} \\ \theta_{k1}^{(l)} \\ \vdots \\ \theta_{kj}^{(l)} \\ \vdots \\ \theta_{kK_l}^{(l)} \end{bmatrix}_{(K_l+1)\times 1} ; \ \bar{\theta_k}^{(l)} = \begin{bmatrix} 0 \\ \theta_{k1}^{(l)} \\ \vdots \\ \theta_{kj}^{(l)} \\ \vdots \\ \theta_{kK_l}^{(l)} \end{bmatrix}$$

where l is from 1 to L-1.

• Net vector $z^{(l)}(i)$ represents all K_l number of net values of i^{th} input data, which is calculated as follows. Note that in this calculation l is from 2 to L since

there are no net nodes at the first layer.

$$z^{(l)}(i) = \begin{bmatrix} z_1^{(l)}(i) \\ z_2^{(l)}(i) \\ \vdots \\ z_k^{(l)}(i) \\ \vdots \\ z_{K_l}^{(l)}(i) \end{bmatrix}_{K_l \times 1} = \theta^{(l-1)} a^{(l-1)}(i)$$

where l is from 2 to L.

$$\begin{aligned} z_k^{(l)}(i) &= \theta_k^{(l-1)\top} a^{(l-1)}(i) \\ &= \theta_{k0}^{(l-1)} a_0^{(l-1)}(i) + \theta_{k1}^{(l-1)} a_1^{(l-1)}(i) + \cdots + \theta_{kK_{l-1}}^{(l-1)} a_{K_{l-1}}^{(l-1)}(i) \\ &= \sum_{j=0}^{K_{l-1}} \theta_{kj}^{(l-1)} a_j^{(l-1)}(i) \in (-\infty, \infty) \end{aligned}$$

• Actual output vector $a^{(l)}(i)$ represents all K_l number of actual output values of i^{th} input data and it is computed as follows. Note that in this calculation l is also from 2 to L since the $a^{(1)}(i)$ is same as $X(i)$.

$$a^{(l)}(i) = \begin{bmatrix} a_1^{(l)}(i) \\ a_2^{(l)}(i) \\ \vdots \\ a_k^{(l)}(i) \\ \vdots \\ a_{K_{(l)}}^{(l)}(i) \end{bmatrix}_{K_{(l)} \times 1} = f(z^{(l)}(i)) \text{ where } f(z_k^{(l)}(i)) = \frac{1}{1+e^{(z_k^{(l)}(i))}}$$

where l is from 2 to L.

$$a_k^{(l)}(i) = f(z_k^{(l)}(i)) \in (0, 1)$$

• Target output vector $y(i)$ represents all K_L number of target output values as follows. Note that each target output value has either 0 or 1.

$$\begin{aligned} y(i) &= \begin{bmatrix} y_1(i) \\ y_2(i) \\ \vdots \\ y_k(i) \\ \vdots \\ y_{K_L}(i) \end{bmatrix}_{K_L \times 1} \\ y_k(i) &\in \{0,\, 1\} \end{aligned}$$

• Error value $\delta_k^{(L)}(i)$ represents the difference between actual output value and target output value at the last layer. Error vector $\delta^{(L)}(i)$ represents all the errors at the last layer and it is computed as follows, which is similar to a 2-layered ANN. Note that the error vectors at the other layers are computed through error update rule which is explained and derived in the following paragraphs.

$$\begin{aligned} \delta_k^{(L)}(i) &= a_k^{(L)}(i) - y_k(i) \in (-1, 1) \\ \delta^{(L)}(i) &= \begin{bmatrix} \delta_1^{(L)}(i) \\ \delta_2^{(L)}(i) \\ \vdots \\ \delta_k^{(L)}(i) \\ \vdots \\ \delta_{K_L}^{(L)}(i) \end{bmatrix}_{K_L \times 1} = a^{(L)}(i) - y(i) \end{aligned}$$

• Error Update. In arbitrary-depth neural network, each layer has each different error values as computed as follows. This error update is necessary to proceed the weight update rule and the Equation (3.12) will be justified in the derivation paragraph.

$$l = L-1 \; to \; 2 \qquad \delta_k^{(l)}(i) = \sum_{v=1}^{K_{l+1}} \theta_{vk}^{(l)} \cdot \delta_v^{(l+1)}(i) \cdot f'(z_k^{(l)}(i)) \tag{3.12}$$

$$\delta^{(l)}(i) = (\theta^{(l)})^\top \delta^{(l+1)}(i) .* f'(z^{(l)}(i))$$

• The cost function $J(\theta)$ in a generalized, arbitrary-depth, ANN also has same role as one in a 2-layered ANN, which gives a way of measuring learning results or learning effects based on the differences between actual and target output values. The difference is that it considers the actual output in the last layer thus the function definition is changed to the Equation (3.13) from Equation (3.6).

$$\begin{aligned} J(\theta) &= \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K_L} (-y_k(i) ln(a_k^{(L)}(i)) - (1-y_k(i)) ln(1-a_k^{(L)}(i))) + \\ &+ \underbrace{\frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{k=1}^{K_{l+1}} \sum_{j=1}^{K_l} (\theta_{kj}^{(l)})^2}_{\text{Regularization Term}} \\ &= \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K_L} Cost_k(i) + R \end{aligned} \tag{3.13}$$

$$\text{where} \begin{cases} Cost_k(i) = (-y_k(i) ln(a_k^{(L)}(i)) - (1-y_k(i)) ln(1-a_k^{(L)}(i))) \\ R = \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{k=1}^{K_{l+1}} \sum_{j=1}^{K_l} (\theta_{kj}^{(l)})^2 \end{cases}$$

Similar to a 2-layered ANN, Figure 3.14 helps us to visually understand how

a generalized, arbitrary-depth, ANN evaluates its cost depending on a given input and target output.

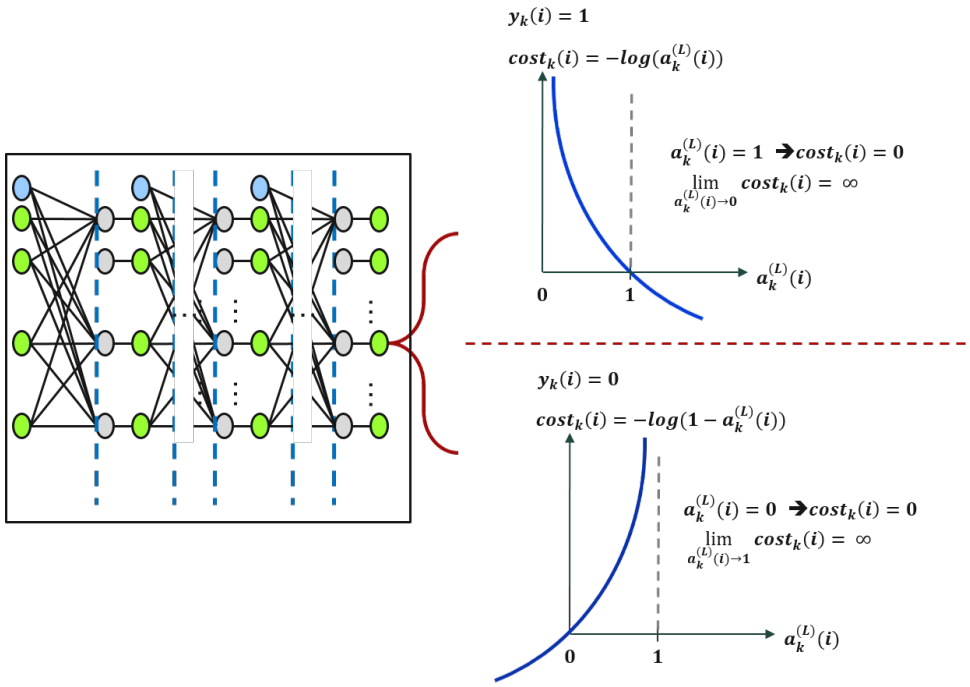


Figure 3.20: Generalized Arbitrary-Depth ANN Role of Cost Function

• Weight Update - Elementwise. Based on the cost function $J(\theta)$ the weight update rule is shown in Equation (3.14). The role of weight update rule is same as the 2-layered neural network except that it is applied to the weights in each layer since it is arbitrary-depth neural network.

$$
\begin{aligned}
& a^{(1)}(i) = X(i) \\
& l = 1 \; to \; L-1 \qquad \theta_{kj}^{(l)}(t+1) = \theta_{kj}^{(l)}(t) - \eta\left(\frac{\partial}{\partial \theta_{kj}^{(l)}} J(\theta)\right) \\
& \qquad\qquad\qquad\qquad = \theta_{kj}^{(l)}(t) - \eta(\Delta\theta_{kj}^{(l)}) \qquad (3.14) \\
& \qquad\qquad \text{where } \Delta\theta_{kj}^{(l)} = \frac{1}{m}\sum_{i=1}^{m} \delta_k^{(l+1)}(i) a_j^{(l)}(i) + \underbrace{\frac{\lambda}{m}\theta_{kj}^{(l)}}_{\text{Regularization Term}} \; (j \geq 1)
\end{aligned}
$$

• Weight Update - Vectorwise. Equation (3.14) can be written in a vectorwise form as following Equation (3.15), which is self-explanatory.

$$
\begin{aligned}
& a^{(1)}(i) = X(i) \\
& l = 1 \; to \; L-1 \qquad \theta_k^{(l)}(t+1) = \theta_k^{(l)}(t) - \eta(\frac{\partial}{\partial \theta_k^{(l)}} J(\theta)) \\
& \qquad\qquad\qquad\qquad = \theta_k^{(l)}(t) - \eta(\Delta\theta_k^{(l)}) \qquad\qquad (3.15) \\
& \qquad\qquad \text{where } \Delta\theta_k^{(l)} = \frac{1}{m}\sum_{i=1}^{m} \delta_k^{(l+1)}(i) a^{(l)}(i) + \frac{\lambda}{m}\bar{\theta_k}^{(l)} \\
& \qquad\qquad\qquad\qquad = \begin{bmatrix} \Delta\theta_{k0}^{(l)} \\ \Delta\theta_{k1}^{(l)} \\ \vdots \\ \Delta\theta_{kj}^{(l)} \\ \vdots \\ \Delta\theta_{kK_l}^{(l)} \end{bmatrix}_{(K_l+1)\times 1} \\
& \qquad\qquad\qquad\qquad = \frac{1}{m}\sum_{i=1}^{m} \delta_k^{(l+1)}(i) \begin{bmatrix} a_0^{(l)}(i) \\ a_1^{(l)}(i) \\ \vdots \\ a_j^{(l)}(i) \\ \vdots \\ a_{K_l}^{(l)}(i) \end{bmatrix}_{(K_l+1)\times 1} + \frac{\lambda}{m}\bar{\theta_k}^{(l)}
\end{aligned}
$$

• Weight Update - matrixwise. The matrixwise weight update rule is shown

Equation (3.16), which is based on Equation (3.15).

$$
\begin{aligned}
& a^{(1)}(i) = X(i) \\
& l = 1 \; to \; L-1 \qquad\qquad \theta^{(l)}(t+1) &&= \theta^{(l)}(t) - \eta(\frac{\Delta}{\Delta\theta^{(l)}} J(\theta)) \\
& &&= \theta^{(l)}(t) - \eta(\Delta\theta^{(l)}) \qquad (3.16) \\
& \text{where } \Delta\theta^{(l)} &&= \frac{1}{m}\sum_{i=1}^{m} \delta^{(l+1)}(i) a^{(l)}(i)^{\top} + \frac{\lambda}{m}\bar{\theta} \\
& \Delta\theta^{(l)} &&= \begin{bmatrix} \Delta\theta_1^{(l)\top} \longrightarrow \\ \Delta\theta_2^{(l)\top} \longrightarrow \\ \vdots \\ \Delta\theta_k^{(l)\top} \longrightarrow \\ \vdots \\ \Delta\theta_{K_{l+1}}^{(l)\top} \longrightarrow \end{bmatrix}_{K_{l+1}\times(K_l+1)}
\end{aligned}
$$

Batch Learning - Matrixwise The Algorithm 6 shows pseudocode for a generalized, arbitrary-depth, ANN with logistic regression and gradient descent optimization on matrixwise batch learning. In this paragraph, we will see how it takes the whole input and target output data at once. Likewise, it is based on vectorwise batch learning and the learning results are same, but the matrixwise computational speed would be faster than vectorwise because there are less for loop.

Algorithm 6 Generalized Arbitrary-depth ANN Batch Learning Matrixwise Pseudocode

Get $X \in R^{m \times n}$, $y \in R^{m \times K}$, $T, m, n, K_1, \cdots, K_l, \cdots, K_L, \eta, \lambda$,
Add $\forall_i, X_0(i) = +1$ for bias $\rightarrow X \in R^{m \times (n+1)}$
for $l = 1$ **to** $L - 1$ **do**
  Init $\theta^{(l)} \in R^{K_{l+1} \times (K_l + 1)}$
end for
for $t = 1$ **to** T **do**
  FORWARD PROPAGATION
  $a^{(1)} = X$
  for l = 2 **to** L **do**
    $z^{(l)} = a^{(l-1)}(\theta^{(l-1)})^\top$
    $a^{(l)} = f(z^{(l)})$
    Add $a_0^{(l)} \leftarrow +1$ for bias at each layer.
  end for
  BACKWARD PROPAGATION
  Remove $a_0^{(L)}$
  $\delta^{(L)} = a^{(L)} - y$
  $\Delta\theta^{(L-1)}(t) = (\delta^{(L)})^\top a^{(L-1)}$
  for $l = L - 1$ **to** 2 **do**
    $\delta^{(l)} = \delta^{(l+1)}\theta^{(l)}$
    Remove $\delta_0^{(l)}$
    $\delta^{(l)} = \delta^{(l)} .* f'(z^{(l)})$
    $\Delta\theta^{(l-1)}(t) = (\delta^{(l)})^\top a^{(l-1)}$
  end for
  WEIGHT UPDATE
  $Cost(t) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K_L} \Big(-y_k(i) ln(a_k^{(L)}(i)) - (1 - y_k(i)) ln(1 - a_k^{(L)}(i)) \Big)$
  for l = 1 **to** L-1 **do**
    $\theta^{(l)}(t+1) = \theta^{(l)}(t) - \eta(\frac{1}{m}\Delta\theta^{(l)}(t) + \bar{\theta}^{(l)}(t))$
  end for
  $R(t) = \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{k=1}^{K_{l+1}} \sum_{j=1}^{K_l} \Big(\theta_{kj}^{(l)} \Big)^2$
  $J(t) = Cost(t) + R(t)$
end for

- Input X is matrix represents all m number of input vectors as follows, which

is same as in vectorwise batch learning.

$$X = \begin{bmatrix} x(1)^\top \longrightarrow \\ x(2)^\top \longrightarrow \\ \vdots \\ x(i)^\top \longrightarrow \\ \vdots \\ x(m)^\top \longrightarrow \end{bmatrix}_{m\times(n+1)} = a^{(1)} = \begin{bmatrix} a^{(1)}(1)^\top \longrightarrow \\ a^{(1)}(2)^\top \longrightarrow \\ \vdots \\ a^{(1)}(i)^\top \longrightarrow \\ \vdots \\ a^{(1)}(m)^\top \longrightarrow \end{bmatrix}_{m\times(K_1+1)}$$

$$X(i) = \begin{bmatrix} x_0(i) \\ x_1(i) \\ \vdots \\ x_j(i) \\ \vdots \\ x_n(i) \end{bmatrix}_{(n+1)\times 1} = a^{(1)}(i) = \begin{bmatrix} a_0^{(1)}(i) \\ a_1^{(1)}(i) \\ \vdots \\ a_j^{(1)}(i) \\ \vdots \\ a_{K_1}^{(1)}(i) \end{bmatrix}_{(K_1+1)\times 1}$$

• Weight matrix $\theta^{(l)}$ and weight vector $\theta_k^{(l)}$ are as follows, which is same as in vectorwise batch learning.

$$\theta^{(l)} = \begin{bmatrix} \theta_1^{(l)\top} \longrightarrow \\ \theta_2^{(l)\top} \longrightarrow \\ \vdots \\ \theta_k^{(l)\top} \longrightarrow \\ \vdots \\ \theta_{K_{l+1}}^{(l)\top} \longrightarrow \end{bmatrix}_{K_{l+1}\times(K_l+1)} ; \ \theta_k^{(l)} = \begin{bmatrix} \theta_{k0}^{(l)} \\ \theta_{k1}^{(l)} \\ \vdots \\ \theta_{kj}^{(l)} \\ \vdots \\ \theta_{kK_l}^{(l)} \end{bmatrix}_{(K_l+1)\times 1} ; \ \bar{\theta_k}^{(l)} = \begin{bmatrix} 0 \\ \theta_{k1}^{(l)} \\ \vdots \\ \theta_{kj}^{(l)} \\ \vdots \\ \theta_{kK_l}^{(l)} \end{bmatrix}$$

where l is from 1 to L-1.

• Net matrix $z^{(l)}$ represents all m number of net vectors at l^{th} layer as follows.

$$z^{(l)} = \begin{bmatrix} z^{(l)}(1)^\top \longrightarrow \\ z^{(l)}(2)^\top \longrightarrow \\ \vdots \\ z^{(l)}(i)^\top \longrightarrow \\ \vdots \\ z^{(l)}(m)^\top \longrightarrow \end{bmatrix}_{m \times K_l} = \begin{bmatrix} a^{(l-1)}(1)(\theta^{(l-1)})^\top \\ a^{(l-1)}(2)(\theta^{(l-1)})^\top \\ \vdots \\ a^{(l-1)}(i)(\theta^{(l-1)})^\top \\ \vdots \\ a^{(l-1)}(m)(\theta^{(l-1)})^\top \end{bmatrix}_{m \times K_l} = a^{(l-1)}(\theta^{(l-1)})^\top$$

$$z^{(l)}(i) = \begin{bmatrix} z_1^{(l)}(i) \\ z_2^{(l)}(i) \\ \vdots \\ z_k^{(l)}(i) \\ \vdots \\ z_{K_l}^{(l)}(i) \end{bmatrix}_{K_l \times 1} = \theta^{(l-1)} a^{(l-1)}(i)$$

• Actual output matrix $a^{(l)}$ represents all m number of actual output vectors at l^{th} layer as follows. f is the same logistic function as in vectorwise batch learning.

$$a^{(l)} = \begin{bmatrix} a^{(l)}(1)^\top \longrightarrow \\ a^{(l)}(2)^\top \longrightarrow \\ \vdots \\ a^{(l)}(i)^\top \longrightarrow \\ \vdots \\ a^{(l)}(m)^\top \longrightarrow \end{bmatrix}_{m \times K_l} = f(z^{(l)})$$

• Target output matrix y represents all m number of given target output vectors as follows.

$$y = \begin{bmatrix} y(1)^\top \longrightarrow \\ y(2)^\top \longrightarrow \\ \vdots \\ y(i)^\top \longrightarrow \\ \vdots \\ y(m)^\top \longrightarrow \end{bmatrix}_{m \times K_L}$$

• Error Update. Error matrix $\delta^{(l)}$ represents all m number of error vectors at l^{th} layer as follows.

$$\begin{aligned} l = L \qquad & \delta^{(L)} = a^{(L)} - y(i) \\ l = L-1 \; to \; 2 \qquad & \delta^{(l)} = \delta^{(l+1)}\theta^{(l)} \\ & = \begin{bmatrix} \delta^{(l+1)}(1)^\top\theta^{(l)} \longrightarrow \\ \delta^{(l+1)}(2)^\top\theta^{(l)} \longrightarrow \\ \vdots \\ \delta^{(l+1)}(i)^\top\theta^{(l)} \longrightarrow \\ \vdots \\ \delta^{(l+1)}(m)^\top\theta^{(l)} \longrightarrow \end{bmatrix}_{m \times K_l} \\ & \delta^{(l)} = \sum_{i=1}^{m}\sum_{k=1}^{K_l}\left(\delta_{ik}^{(l)} f'(z_{ik}^{(l)})\right) \end{aligned} \tag{3.17}$$

• The cost function for a generalized, arbitrary-depth, ANN on matrixwise batch learning is defined as follows. The results of this is same as the one in vectorwise

learning but the calculation is different since it handles changed mathematical forms.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K_L}\left(-y_{ik}ln(a_{ik}^{(L)}) - (1-y_{ik})ln(1-a_{ik}^{(L)})\right) + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{k=1}^{K_{l+1}}\sum_{j=1}^{K_l}\left(\theta_{kj}^{(l)}\right)^2 \tag{3.18}$$

• Weight Update for matrixwise batch learning is as follows.

$$a^{(1)} = X$$

$$l = 1\ to\ L-1 \quad \theta^{(l)}(t+1) = \theta^{(l)}(t) - \eta\left(\frac{1}{m}\delta^{(l+1)\top}a^{(l)} + \frac{\lambda}{m}\bar{\theta}^{(l)}\right) \tag{3.19}$$

$$\left(= \theta^{(l)}(t) - \eta\left(\frac{1}{m}\sum_{i=1}^{m}\delta^{(l+1)}(i)a^{(l)}(i)^\top + \frac{\lambda}{m}\bar{\theta}^{(l)}\right)\right)$$

Derivation of Weight Update Rule and Error Update Rule This paragraph derives the Equation (3.14) and 3.12. The mathematical process for the derivation fully described each step by step to be easily understood as follows.

$l = 1 to L-1$

$$\begin{aligned}
\theta_{kj}^{(l)}(t+1) &= \theta_{kj}^{(l)}(t) - \eta(\tfrac{\partial}{\partial\theta_{kj}^{(l)}}J(\theta)) \\
&= \theta_{kj}^{(l)}(t) - \eta(\Delta\theta_{kj}^{(l)}) \\
\Delta\theta_{kj}^{(l)} &= \tfrac{\partial}{\partial\theta_{kj}^{(l)}}J(\theta) \\
&= \tfrac{\partial}{\partial\theta_{kj}^{(l)}}(\tfrac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K_L}(-y_k(i)ln(a_k^{(L)}(i)) - (1-y_k(i))ln(1-a_k^{(L)}(i))) \\
&+ \tfrac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{k=1}^{K_{l+1}}\sum_{j=1}^{K_l}(\theta_{kj}^{(l)})^2) \\
&= \tfrac{\partial}{\partial\theta_{kj}^{(l)}}(\tfrac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K_L}Cost_k(i) + R)
\end{aligned}$$

$l = L - 1$

$$\Delta\theta_{kj}^{(L-1)} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial\theta_{kj}^{(L-1)}}(\sum_{k=1}^{K_L} Cost_k(i)) + \frac{\partial}{\partial\theta_{kj}^{(L-1)}}R$$

$k = 1 to K_L;$

$j = 0 to K_{L-1};$

$$\begin{aligned}\frac{\partial}{\partial\theta_{kj}^{(L-1)}}(\sum_{k=1}^{K_L} Cost_k(i)) &= \frac{\partial}{\partial\theta_{kj}^{(L-1)}}Cost_k(i)\\ &= \frac{\partial}{\partial\theta_{kj}^{(L-1)}}(-y_k(i)ln(a_k^{(L)}(i)) - (1-y_k(i))ln(1-a_k^{(L)}(i)))\\ &= \frac{\partial}{\partial a_k^{(L)}(i)}Cost_k(i)\cdot\frac{\partial a_k^{(L)}(i)}{\partial z_k^{(L)}(i)}\cdot\frac{\partial z_k^{(L)}(i)}{\partial\theta_{kj}^{(L-1)}}\end{aligned}$$

$$\begin{aligned}\frac{\partial}{\partial a_k^{(L)}(i)}Cost_k(i) &= \frac{-y_k(i)}{a_k^{(L)}(i)} - \frac{1-y_k(i)}{1-a_k^{(L)}(i)}\cdot(1-a_k^{(L)}(i))'(\because ln(x)' = \frac{1}{x})\\ &= \frac{-y_k(i)}{a_k^{(L)}(i)} + \frac{1-y_k(i)}{1-a_k^{(L)}(i)}\\ &= \frac{a_k^{(L)}(i) - a_k^{(L)}(i)\cdot y_k(i) - y_k(i) + a_k^{(L)}(i)\cdot y_k(i)}{a_k^{(L)}(i)(1-a_k^{(L)}(i))}\\ &= \frac{a_k^{(L)}(i) - y_k(i)}{a_k^{(L)}(i)(1-a_k^{(L)}(i))}\\ &= \frac{\delta_k^{(L)}(i)}{a_k^{(L)}(i)(1-a_k^{(L)}(i))}\end{aligned}$$

$$\begin{aligned}\frac{\partial a_k^{(L)}(i)}{\partial z_k^{(L)}(i)} &= \frac{\partial f(z_k^{(L)}(i))}{\partial z_k^{(L)}(i)}\\ &= f(z_k^{(L)}(i))(1-f(z_k^{(L)}(i)))\\ &(\because f(x) = \frac{1}{1+e^{-x}}, f'(x) = f(x)(1-f(x)))\\ &= a_k^{(L)}(i)(1-a_k^{(L)}(i))\end{aligned}$$

$$\begin{aligned}\frac{\partial z_k^{(L)}(i)}{\partial\theta_{kj}^{(L-1)}} &= \frac{\partial}{\partial\theta_{kj}^{(L-1)}}(\sum_{j=1}^{K_{L-1}}\theta_{kj}^{(L-1)}a_j^{(L-1)}(i))\\ &= a_j^{(L-1)}(i)\end{aligned}$$

$$
\begin{aligned}
\therefore \frac{\partial}{\partial \theta_{kj}^{(L-1)}} Cost_k(i) &= \frac{\delta_k^{(L)}(i)}{a_k^{(L)}(i)(1-a_k^{(L)}(i))} \cdot \frac{a_k^{(L)}(i)(1-a_k^{(L)}(i))}{1} \cdot a_j^{(L-1)}(i) \\
&= \delta_k^{(L)}(i) \cdot a_j^{(L-1)}(i) \\
\frac{\partial}{\partial \theta_{kj}^{(L-1)}} R &= \frac{\partial}{\partial \theta_{kj}^{(L-1)}} (\frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{k=1}^{K_{l+1}} \sum_{j=1}^{K_l} (\theta_{kj}^{(l)})^2) \\
&= \frac{\lambda}{m} \theta_{kj\ (j \geq 1)}^{(L-1)} \\
\therefore \Delta \theta_{kj}^{(L-1)} &= \frac{1}{m} \sum_{i=1}^{m} \delta_k^{(L)}(i) \cdot a_j^{(L-1)}(i) + \frac{\lambda}{m} \theta_{kj\ (j \geq 1)}^{(L-1)} \\
\therefore \theta_{kj}^{(L-1)}(t+1) &= \theta_{kj}^{(L-1)}(t) - \eta(\frac{1}{m} \sum_{i=1}^{m} \delta_k^{(L)}(i) \cdot a_j^{(L-1)}(i) + \frac{\lambda}{m} \theta_{kj\ (j \geq 1)}^{(L-1)})
\end{aligned}
$$

$l = L-2$

$$\Delta\theta_{kj}^{(L-2)} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial\theta_{kj}^{(L-2)}}(\sum_{v=1}^{K_L}Cost_v(i)) + \frac{\partial}{\partial\theta_{kj}^{(L-2)}}R$$

$v = 1toK_L$;

$k = 1toK_{L-1}$;

$j = 0toK_{L-2}$;

$$\begin{aligned}
\frac{\partial}{\partial\theta_{kj}^{(L-2)}}(\sum_{v=1}^{K_L}Cost_v(i)) &= \sum_{v=1}^{K_L}(\frac{\partial}{\partial\theta_{kj}^{(L-2)}}Cost_v(i)) \\
&= \sum_{v=1}^{K_L}(\frac{\partial}{\partial\theta_{kj}^{(L-2)}}(-y_v(i)ln(a_v^{(L)}(i)) - (1-y_v(i))ln(1-a_v^{(L)}(i)))) \\
&= \sum_{v=1}^{K_L}(\frac{\partial}{\partial a_v^{(L)}(i)}Cost_v(i)\cdot\frac{\partial a_v^{(L)}(i)}{\partial z_v^{(L)}(i)}\cdot\frac{\partial z_v^{(L)}(i)}{\partial\theta_{kj}^{(L-2)}}) \\
&= \sum_{v=1}^{K_L}(\frac{(a_v^{(L)}-y_v)}{(a_v^{(L)}(1-(a_v^{(L)})}\cdot\frac{(a_v^{(L)}(1-(a_v^{(L)})}{1}\cdot\frac{\partial}{\partial\theta_{kj}^{(L-2)}}(\sum_{k=0}^{K_{L-1}}\theta_{vk}^{(L-1)}a_k^{(L-1)})) \\
&= \sum_{v=1}^{K_L}(\delta_v^{(L)}(i)\cdot\theta_{vk}^{(L-1)}\cdot\frac{\partial}{\partial\theta_{kj}^{(L-2)}}(\sum_{k=0}^{K_{L-1}}a_k^{(L-1)})) \\
&= \sum_{v=1}^{K_L}(\delta_v^{(L)}(i)\cdot\theta_{vk}^{(L-1)}\cdot\frac{\partial}{\partial\theta_{kj}^{(L-2)}}(\sum_{k=0}^{K_{L-1}}f(z_k^{(L-1)}))) \\
&= \sum_{v=1}^{K_L}(\delta_v^{(L)}(i)\cdot\theta_{vk}^{(L-1)}\cdot(\sum_{k=0}^{K_{L-1}}f'(z_k^{(L-1)})\cdot\frac{\partial}{\partial\theta_{kj}^{(L-2)}}z_k^{(L-1)})) \\
&= \sum_{v=1}^{K_L}(\delta_v^{(L)}(i)\cdot\theta_{vk}^{(L-1)}\cdot f'(z_k^{(L-1)})\cdot\frac{\partial}{\partial\theta_{kj}^{(L-2)}}z_k^{(L-1)}) \\
&= \sum_{v=1}^{K_L}(\delta_v^{(L)}(i)\cdot\theta_{vk}^{(L-1)}\cdot f'(z_k^{(L-1)})\cdot\frac{\partial}{\partial\theta_{kj}^{(L-2)}}(\sum_{j=0}^{K_{L-2}}\theta_{kj}^{(L-2)}a_j^{(L-2)})) \\
&= \sum_{v=1}^{K_L}(\delta_v^{(L)}(i)\cdot\theta_{vk}^{(L-1)}\cdot f'(z_k^{(L-1)})\cdot a_j^{(L-2)}) \\
&= \delta_k^{(L-1)}\cdot a_j^{(L-2)}
\end{aligned}$$

where $\delta_k^{(L-1)} = \sum_{v=1}^{K_L}(\delta_v^{(L)}(i)\cdot\theta_{vk}^{(L-1)})\cdot f'(z_k^{(L-1)})$

$$
\begin{aligned}
\tfrac{\partial}{\partial\theta_{kj}^{(L-2)}}R \quad &= \tfrac{\partial}{\partial\theta_{kj}^{(L-2)}}(\tfrac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{k=1}^{K_{l+1}}\sum_{j=1}^{K_l}(\theta_{kj}^{(l)})^2) \\
&= \tfrac{\lambda}{m}\theta_{kj\ (j\geq 1)}^{(L-2)} \\
\therefore \Delta\theta_{kj}^{(L-2)} \quad &= \delta_k^{(L-1)}\cdot a_j^{(L-2)} \\
&\quad \text{where } \delta_k^{(L-1)} = \sum_{v=1}^{K_L}(\delta_v^{(L)}(i)\cdot\theta_{vk}^{(L-1)})\cdot f'(z_k^{(L-1)}) \\
\therefore \theta_{kj}^{(L-2)}(t+1) \quad &= \theta_{kj}^{(L-2)}(t) - \eta(\delta_k^{(L-1)}\cdot a_j^{(L-2)} + \tfrac{\lambda}{m}\theta_{kj\ (j\geq 1)}^{(L-2)}) \\
&\quad \text{where } \delta_k^{(L-1)} = \sum_{v=1}^{K_L}(\delta_v^{(L)}(i)\cdot\theta_{vk}^{(L-1)})\cdot f'(z_k^{(L-1)})
\end{aligned}
$$

Therefore,

$l = L$

$$\delta_k^{(L)} = a_k^{(L)} - y_k$$

$l = (L-1)to1$

$$
\begin{aligned}
\delta_k^{(l)} \quad &= \sum_{v=1}^{K_{l+1}}\theta_{vk}^{(l)}\cdot\delta_v^{(l+1)}(i)\cdot f'(z_k^{(l)}(i)) \\
\theta_{kj}^{(l)}(t+1) \quad &= \theta_{kj}^{(l)}(t) - \eta(\tfrac{1}{m}\sum_{i=1}^{m}\delta_k^{(l+1)}(i)a_j^{(l)}(i) + \tfrac{\lambda}{m}\theta_{kj\ (j\geq 1)}^{(l)})
\end{aligned}
$$

3.3 Reward-based Neural Model

3.3.1 Hebbian Plasticity

In 1949, Donald Hebb introduced the Hebbian plasticity which covers the relationship between output of presynaptic and postsynaptic neurons [16]. The Hebbian plasticity is based on the discovery, "when a presynaptic neuron repeatedly participates in firing of a postsynaptic neuron, the strength between pre- and postsynaptic neurons increases", which can be simply represented by him as "neurons fire together wire together".

An example is Pavlov's famous "conditioned reflexes" experiments with dogs [36]. A dog salivates when food is presented. The food can be seen as an unconditioned stimulus for the dog's response—salivating. The Hebbian learning rule uses this existing relationship between the unconditioned stimulus and the dog's response in the learning process. For example, assume we set all weights to positive values for the relationship between the stimulus of food and the response of salivating (that is, the synaptic weights between the pre-synaptic neurons that respond to the food sensation and the post-synaptic neurons that generate salivation are greater than zero) and set the weights to neutral values for the relationship between a new stimulus, such as the sound of ringing a bell, and the response of salivating. Then, after calculating the output of the neurons, the response is still positive, which means the dog will salivate and the relationship between ringing the bell and salivating will be updated according to Equation (3.20) to positive values from neutral values—that is, the dog will have learned to salivate at the sound of the bell. The detail computational process is described in next learning paragraph.

Iterative Hebbian Learning Algorithm 7 shows the pseudocode of iterative Hebbian learning. Note that iterative learning refers to take one input data example in one learning process whereas batch learning computes all m number of input data examples in updating weight vector (s) through one learning step. Hebbian learning is typically based on 2-layered ANN thus the following mathematical process in Chapter 3.3 is based on 2-layered ANN. However, in this dissertation, Hebbian learning does not consider the bias nodes thus the Hebbian ANN architecture is seen by removing all bias nodes and corresponding

links in the Figure 3.9.

Algorithm 7 Iterative Hebbian Learning Pseudocode

Set T, η
Init $\theta \in R^{K \times n}$
for $t = 1$ **to** T **do**
  Get $X(t) \in R^n$
  $z(t) = \theta X(t)$
  $a(t) = f(z(t))$,where $f(z) = z$
  $\Delta\theta(t) = \eta(a(t)X(t)^\top)$
  $\theta(t+1) = \theta(t) + \Delta\theta(t)$
  $\theta = \theta(t+1)$
end for

• Input vector $X(t)$ represents one input vector at learning step t. Note that the input dimension is exactly n, not $n+1$, since there is no bias in Hebbian ANN; also there is no i which was indicator each input data example of all m number of input since Hebbian learning takes only one input in one learning process iteratively.

$$X(t) = \begin{bmatrix} x_1(t) \\ \vdots \\ x_j(t) \\ \vdots \\ x_n(t) \end{bmatrix}_{(n)\times 1}$$

• Weight matrix θ and weight vector θ_k as follows. Note that θ refers to all

weights in the ANN and θ_k indicates each weight vector of it.

$$\theta = \begin{bmatrix} \theta_1^\top \longrightarrow \\ \theta_2^\top \longrightarrow \\ \vdots \\ \theta_k^\top \longrightarrow \\ \vdots \\ \theta_K^\top \longrightarrow \end{bmatrix}_{K\times(n)} ; \ \theta_k = \begin{bmatrix} \theta_{k1} \\ \vdots \\ \theta_{kj} \\ \vdots \\ \theta_{kn} \end{bmatrix}_{(n)\times 1} ; \ \bar{\theta_k} = \begin{bmatrix} 0 \\ \theta_{k1} \\ \vdots \\ \theta_{kj} \\ \vdots \\ \theta_{kn} \end{bmatrix} \text{ where } \theta_{k0} = 0$$

• Net vector $z(t)$ represents K number of net values at learning step t, which is calculated as follows.

$$\begin{aligned} z(t) &= \begin{bmatrix} z_1(t) \\ z_2(t) \\ \vdots \\ z_k(t) \\ \vdots \\ z_K(t) \end{bmatrix}_{K\times 1} = \theta X(t) \\ z_k(t) &= \theta_k^\top X(t) \\ &= \theta_{k1}x_1(t) + \cdots + \theta_{kn}x_n(t) \\ &= \sum_{j=1}^{n} \theta_{kj}x_j(t) \in (-\infty, \infty) \end{aligned}$$

• Actual output vector $a(t)$ represents K number of actual output values at learning step t, which is computed as follows. Note that the activation function used in Hebbian learning is simple linear function: $f(x) = x$. This means that

value of an actual output node is same as the corresponding net value.

$$
\begin{aligned}
a(t) &= \begin{bmatrix} a_1(t) \\ a_2(t) \\ \vdots \\ a_k(t) \\ \vdots \\ a_K(t) \end{bmatrix}_{K\times 1} = f(z(t)) \text{ where } f(z_k(t)) = z_k(t) \\
a_k(t) &= f(z_k(t)) \in (-\infty, \infty)
\end{aligned}
$$

• Weight Update - Elementwise. Equation (3.20) shows how each weight is updated in Hebbian learning, which is also called Hebb's rule [16]. In Hebb's rule, weight update is computed by simple product of presynaptic and postsynaptic neuron's membrane potential like in Equation (3.20). This means the strength of the input and output potentials are correlated each other in determining the synaptic connection which is wight.

$$
\begin{aligned}
\theta_{kj}(t+1) &= \theta_{kj}(t) + \eta(\Delta\theta_{kj}) \\
\text{where } \Delta\theta_{kj} &= \eta(a_k(t) \cdot x_j(t))
\end{aligned}
\tag{3.20}
$$

• Weight Update - Vectorwise. Equation (3.20) can be re-written in a vectorwise

form as following Equation (3.21), which is self-explanatory.

$$\begin{aligned} &\theta_k(t+1) = \theta_k(t) + \eta(\Delta\theta_k) \\ &\text{where } \Delta\theta_k = \eta(a_k(t)X(t)) \\ &\Delta\theta_k = \begin{bmatrix} \Delta\theta_{k1} \\ \vdots \\ \Delta\theta_{kj} \\ \vdots \\ \Delta\theta_{kn} \end{bmatrix}_{(n)\times 1} = a_k(t) \begin{bmatrix} x_1(t) \\ \vdots \\ x_j(t) \\ \vdots \\ x_n(t) \end{bmatrix}_{(n)\times 1} = a_k(t)X(t) \end{aligned} \tag{3.21}$$

Looking deep into the Equation (3.21), we can see the role of Hebbian weight update. The Equation (3.21) can be simplified as follows by assuming η as 1.

Assume: $\eta = 1$ and α = angular distance between θ_k and $X(t)$

Then: $\theta_k(t+1) = \theta_k(t) + a_k(t)X(t)$

In this simplified equation, θ_k can be considered as a k^{th} learning vector corresponding to a_k, which decides its behavior based on the current relationship between θ_k and input vector $X(t)$. We can understand its behavior by looking into three possible cases: (1) $a_k(t) = 0$, (2) $a_k(t) > 0$, and (3) $a_k(t) < 0$. First, in the case of $a_k(t) = 0$, the weight vector θ_k has no changes in the weight update as follows.

$$\begin{cases} a_k(t) = 0 & \iff \theta_k^\top X(t) = 0 \\ & \iff \theta_k \cdot X(t) = 0 \ (\|\theta_k\|\|X(t)\| \cos(\alpha) = 0) \\ & \implies \theta_k \text{ and } X(t) \text{ are perpendicular or parallel} \\ & \implies \theta_k(t+1) = \theta_k(t) \end{cases}$$

Second, if $a_k(t) > 0$, we can see it ultimately makes the weight vector θ_k closer to the current input vector $X(t)$ as follows.

$$\begin{cases} a_k(t) > 0 & \iff \theta_k^\top X(t) > 0 \\ & \iff \theta_k \cdot X(t) > 0 \ (\|\theta_k\|\|X(t)\| \cos(\alpha) > 0) \\ & \implies 0 < \alpha < (\frac{\pi}{2}) \text{ or } (\frac{3\pi}{2}) < \alpha < (\frac{4\pi}{2}) \\ & \implies \theta_k \text{ and } X(t) \text{ have closer angular distance} \\ & \implies \theta_k(t+1) = \theta_k(t) + a_k(t)X(t) \\ & \implies \theta_k(t+1) \text{ will be closer to } X(t) \end{cases}$$

Third, in the case of $a_k(t) > 0$, the weight vector θ_k is farther away from the current input vector $X(t)$ as follows.

$$
\begin{cases}
a_k(t) < 0 & \iff \theta_k^\top X(t) < 0 \\
& \iff \theta_k \cdot X(t) < 0 \ (\|\theta_k\| \|X(t)\| \cos(\alpha) < 0) \\
& \implies (\frac{\pi}{2}) < \alpha < (\frac{3\pi}{2}) \\
& \implies \theta_k \text{ and } X(t) \text{ have far angular distance} \\
& \implies \theta_k(t+1) = \theta_k(t) - a_k(t) X(t) \\
& \implies \theta_k(t+1) \text{ will be further away from } X(t)
\end{cases}
$$

Based on the three cases, we can infer the important characteristics of the Hebbian plasticity: (1) if their angular distance between θ_k and $X(t)$ is close enough to have positive inner product value, they will be closer as a result of learning process, (2) if the relationship is far enough to take negative inner product value, they will further away, and (3) if the relationship between θ_k and input vector $X(t)$ is perpendicular or parallel, θ_k will be stay on same position which means there will be no learning effects. This implies that the Hebbian rule makes each θ_k will keep moving/adjusting either closer to or farther away from $X(t)$ based on their relationships until they become perpendicular or parallel.

• Weight Update - matrixwise. Equation (3.21) can be re-written in a matrixwise

form as following Equation (3.22).

$$\begin{aligned} \theta(t+1) &= \theta(t) + \eta(\Delta\theta) \\ \text{where } \Delta\theta &= \eta(a(t)X(t)^\top) \\ \Delta\theta &= \begin{bmatrix} \Delta\theta_1^\top \longrightarrow \\ \Delta\theta_2^\top \longrightarrow \\ \vdots \\ \Delta\theta_k^\top \longrightarrow \\ \vdots \\ \Delta\theta_K^\top \longrightarrow \end{bmatrix}_{K\times(n)} = a(t)X(t)^\top \end{aligned} \tag{3.22}$$

3.3.2 Reward-based Hebbian Plasticity

Reward-based Hebbian plasticity is based on the Hebbian plasticity, which adds reward value from outside sources such as an environment, a trainer, or chemical processes. As covered in Chapter 2, it is known that additional chemical signals affect synaptic changes, which is the basis of a modulated Hebbian model [38] [53]. In neurobiological perspective, neurotransmitters play important role in changing synaptic plasticity and some of them are related to involving reward information so that the short-lived synaptic association between presynaptic and postsynaptic neuron can be activated in longer time. In this case, modulatory Hebbian learning is kind of reward-based Hebbian learning where reward value is applied in the form of a modulatory signal which turns into a numerical value either $+1$ or -1. The computational process is described in the next learning paragraph.

Iterative Reward-based Hebbian Learning Algorithm 8 shows the pseudocode for iterative reward-based Hebbian learning. Note that the computational part for input vector ($X(t)$), weight matrix (θ), net vector ($z(t)$), and actual output vector ($a(t)$) are exactly same as in Hebbian learning and thus this paragraph only covers how the reward value is applied in adjusting weights.

Algorithm 8 Iterative Reward-based Hebbian Learning Pseudocode

Set T, η
Init $\theta \in R^{K \times n}$
for $t = 1$ **to** T **do**
  Get $X(t) \in R^n$
  $z(t) = \theta X(t)$
  $a(t) = f(z(t))$,where $f(z) = z$
  Get $r(t) \in R^K$
  $\Delta\theta(t) = \eta(a(t). * r(t))X(t)^\top$
  $\theta(t+1) = \theta(t) + \Delta\theta(t)$
  $\theta = \theta(t+1)$
end for

• Reward vector $r(t)$ is generated vector based on reward value $reward(t)$. Reward-based Hebbian learning is taking reward value at each learning step which can be acquired from outside of the learning process like a modulatory signal. In other words, after selecting the actual output node which has maximum value among all of them, reward-based Hebbian ANN waits for a reward and the reward is corresponding only to the selected actual output node. This implies important characteristic that reward-based Hebbian learning is selective learning, which means it updates only selected weights which are associated with the selected actual output based on the reward value. Therefore the weights corresponding to the non-selected actual output nodes should not be affected by the reward value. In this case, there is new computational component which is

called reward vector $r(t)$ and it is set as follows based on a reward value.

$$\begin{aligned}
\text{reward(t)} &\in \{+1, -1\} \text{ given from outside at learning step } t \\
r(t) &= \begin{bmatrix} r_1(t) \\ r_2(t) \\ \vdots \\ r_k(t) \\ \vdots \\ r_K(t) \end{bmatrix}_{K\times 1} \\
r_k(t) &= reward(t); \text{ if } a_k \text{ is selected output at learning step } t \\
r_k(t) &= 0; \text{ if } a_k \text{ is non-selected output at learning step } t
\end{aligned}$$

Note that reward value $reward(t)$ has either $+1$ or -1 and this value is only set to be $r_k(t)$ where k is the selected actual output index; the other elements of the reward vector is set to zeros so as not to have no learning effects.

• Weight Update - Elementwise. Equation (3.23) shows how each weight is updated in reward-based Hebbian learning, which is elementwise weight update rule.

$$\begin{aligned}
\theta_{kj}(i+1) &= \theta_{kj}(t) + \eta(\Delta\theta_{kj}) \\
\text{where } \Delta\theta_{kj} &= r_k(t) \cdot a_k(t) \cdot x_j(t)
\end{aligned} \tag{3.23}$$

• Weight Update - Vectorwise. Equation (3.23) can be re-written in a vectorwise

form as follows, which is self-explanatory.

$$\theta_k(i+1) = \theta(t) + \eta(\Delta\theta_k) \tag{3.24}$$

where $\Delta\theta_k = (r_k(t). * a_k(t))X(t)$

$$\Delta\theta_k = \begin{bmatrix} \Delta\theta_{k1} \\ \vdots \\ \Delta\theta_{kj} \\ \vdots \\ \Delta\theta_{kn} \end{bmatrix}_{(n)\times 1} = r_k(t) \cdot a_k(t) \begin{bmatrix} x_1(t) \\ \vdots \\ x_j(t) \\ \vdots \\ x_n(t) \end{bmatrix}_{(n)\times 1} = r_k(t) \cdot a_k(t) \cdot X(t)$$

Looking deep into the Equation (3.24), we can see the role of reward-based Hebbian weight update. The Equation (3.24) can be simplified as follows by assuming η as 1.

Assume: $\eta = 1$ and $\alpha =$ angular distance between θ_k and $X(t)$

Then: $\theta(t+1)_k = \theta(t) + ((r_k(t) \cdot a_k(t))X(t))$

In this simplified equation, $r_k(t)$ plays an important role in deciding the behavior of each weight vector. Note that reward value is either $+1$ or -1 but $r_k(t)$ is one among $\{-1, 0, +1\}$. Similar to the role of Hebbian learning, there are three cases of $a_k(t)$: (1) $a_k(t) = 0$, (2) $a_k(t) > 0$, and (3) $a_k(t) < 0$; and we can see how $r_k(t)$ affects the learning results in each case. First, in the case of $a_k(t) = 0$, the weight vector θ_k has no changes in the weight update rule as follows.

$$
\begin{cases}
a_k(t) = 0 & \iff \theta_k^\top X(t) = 0 \\
& \iff \theta_k \cdot X(t) = 0 \ (\|\theta_k\| \|X(t)\| \cos(\alpha) = 0) \\
& \implies \Delta\theta_k = 0 \\
& \implies \theta_k(t+1) = \theta_k(t)
\end{cases}
$$

Second, in the case of $a_k(t) > 0$ or $a_k(t) < 0$, the weight vector θ_k is updated based on $r_k(t)$ as follows.

$$
\begin{cases}
a_k(t) > 0 & \iff \theta_k^\top X(t) > 0 \\
& \iff \theta_k \cdot X(t) > 0 (\|\theta_k\| \|X(t)\| \cos(\alpha) > 0) \\
& \implies 0 < \alpha < (\frac{\pi}{2}) \text{ or } (\frac{3\pi}{2}) < \alpha < (\frac{4\pi}{2}) \\
& \implies \theta_k \text{ and } X(t) \text{ have closer angular distance} \\
& \implies \Delta\theta_k = +(r_k(t)\beta X(t) \text{for some positive } \beta \\
& \text{If } r_k(t) = 0 \\
& \implies \theta_k(t+1) = \theta_k(t) \\
& \text{If } r_k(t) = +1 \\
& \implies \theta_k(i+1) = \theta_k(t) + \beta X(t) \\
& \implies \theta_k(i+1) \text{ will be closer to } X(t) \\
& \text{If } r_k(t) = -1 \\
& \implies \theta_k(i+1) = \theta_k(t) - \beta X(t) \\
& \implies \theta_k(i+1) \text{ will be are away from } X(t)
\end{cases}
$$

$$
\begin{cases}
a_k(t) < 0 & \iff \theta_k^\top X(t) < 0 \\
& \iff \theta_k \cdot X(t) < 0 (\|\theta_k\| \|X(t)\| \cos(\alpha) < 0) \\
& \implies (\frac{\pi}{2}) < \alpha < (\frac{3\pi}{2}) \\
& \implies \theta_k \text{ and } X(t) \text{ have far angular distance} \\
& \implies \Delta\theta_k = -(r_k(t)\beta X(t) \text{for some positive } \beta \\
& \text{If } r_k(t) = 0 \\
& \implies \theta_k(t+1) = \theta_k(t) \\
& \text{If } r_k(t) = +1 \\
& \implies \theta_k(i+1) = \theta_k(t) - \beta X(t) \\
& \implies \theta_k(i+1) \text{ will be are away from } X(t) \\
& \text{If } r_k(t) = -1 \\
& \implies \theta_k(i+1) = \theta_k(t) + \beta X(t) \\
& \implies \theta_k(i+1) \text{ will be closer to } X(t)
\end{cases}
$$

Based on the three cases, we can infer important characteristics of the reward-based Hebbian plasticity: (1) if positive reward value, the associated weight vector reinforces the current relationship between itself and input vector by either increasing or decreasing its angular distance, (2) if negative reward value is assigned, the associated weight vector weaken its original relationship between itself and the input vector, and (3) if a weight vector is not associated with the selected actual output node, it is not subject to be learned.

• Weight Update - matrixwise. Equation (3.24) can be re-written in a matrixwise form as following Equation (3.25).

$$\theta(i+1) = \theta(t) + \eta(\Delta\theta) \tag{3.25}$$

$$\text{where } \Delta\theta = (r(t). * a(t))X(t)^\top$$

$$\Delta\theta = \begin{bmatrix} \Delta\theta_1^\top \longrightarrow \\ \Delta\theta_2^\top \longrightarrow \\ \vdots \\ \Delta\theta_k^\top \longrightarrow \\ \vdots \\ \Delta\theta_K^\top \longrightarrow \end{bmatrix}_{K\times(n)} = (r(t). * a(t))X(t)^\top$$

3.3.3 Reward-based Hyperbolic Hebbian Plasticity

Reward-based hyperbolic Hebbian Learning is mostly similar to the reward-based Hebbian learning except that the actual output is given through the hyperbolic tangent activation function. In Hebbian and reward-based Hebbian learning, the value of each actual output node can be increased or decreased infinitely; reward-based hyperbolic Hebbian learning can avoid this infinite boundaries by having the hyperbolic activation function. This idea is introduced first by Soltoggio and Stanley and the computational process is described in a sound way with 2 layered domain-specific neural networks [53]. Based on their work, this section introduced the generalized version of reward-based Hyperbolic Hebbian learning.

Iterative Reward-based Hyperbolic Hebbian Learning Algorithm 9 is the pseudocode for iterative reward-based hyperbolic Hebbian learning. The general computational process including weight update rules and the role of the weight update are same as the reward-based Hebbian learning; thus this paragraph covers how the actual output is calculated.

Algorithm 9 Iterative Reward-based Hyperbolic Hebbian Learning Pseudocode

Set T, η
No bias in Hebbian Plasticity.
Init $\theta \in R^{K \times n}$
for $t = 1$ **to** T **do**
  Get $X(t) \in R^n$
  $z(t) = \theta X(t)$
  $a(t) = f(z(t))$,where $f(z) = \tanh(z)$
  Get $r(t) \in R^K$
  $\Delta\theta(t) = \eta(a(t). * r(t))X(t)^\top$
  $\theta(i+1) = \theta(t) + \Delta\theta(t)$
  $\theta = \theta(i+1)$
end for

• Actual output vector $a(t)$ represents K number of actual output nodes at learning step t, which is computed as follows. Note that the activation function is hyperbolic tangent function. By introducing this hyperbolic tangent function, the actual output can have its maximum and minimum boundary and thus it can avoid indefinite growth of weights and actual output value in the learning

process. Also note that the result of hyperbolic function is between -1 and 1.

$$
\begin{aligned}
a(t) &= \begin{bmatrix} a_1(t) \\ a_2(t) \\ \vdots \\ a_k(t) \\ \vdots \\ a_K(t) \end{bmatrix}_{K\times 1} = f(z(t)) \text{ where } f(z_k(t)) = \tanh(z_k(t)) \\
a_k(t) &= f(z_k(t)) \in (-1, 1)
\end{aligned}
$$

Chapter 4

Related Works

This chapter provides the preceding works related to this research, which are categorized into reward-based, neurorobotics, and context-based robot learning. This related works are also based on the background knowledge introduced in the Chapter 2 and Chapter 3.

4.1 Reward-based Learning

Reward-based learning models have been investigated through several approaches such as Hebbian plasticity or Spike Timing-Dependent Plasticity (STDP). First, as we discovered in Chapter 3.3, Hebbian learning is based on the simple correlation of input and output signals. The learning process is usually performed in 2-layered neural network and directly applies reward value from an environment into adjusting the neural weights, which is either increasing or decreasing an selective weight strength. Based on this simple correlation, there have been noticed several limitations of the basic Hebbian plasticity. First, if input signal is strong, it causes output to be strong and the increased output also makes the synaptic connection strengthen; thus in the simple neural model, firing from pre-synapse and post-synapse can cause indefinite weight growth. Once the weight growth exceeds a maximum boundary, this can make it hard to understand the further connections between the over-calculated synaptic strength and the output. The second limitation is that there is no long-term

potentiation (LTP) in simple Hebbian learning. This means calculating output is based on a short-lived response so the learning can be only performed in a short time without considering the long-term relationship between the wired two neurons. The third limitation is there is no way to decrease the strengthened weight value. In this regards, this section introduces several preceding research works on building novel learning models which are inspired by the basic Hebbian learning. Pennartz develops the *Hebbian synapses with adaptive thresholds* (HSAT) model by combining supervised and Hebbian learning [38]. The main difference between this model and simple Hebbian learning is to make the model affected by modulatory learning with a reward-processing module (RPM) and using errors between input and output from 3-layered neural network. RPM is considered as one special neural node in the network, which regulates the release of calcium (Ca^{+2}) based on reward value; increased amount of the calcium strengthens the neural weights or the decreased one weakens the weights. Pennartz used three types of experiment to demonstrate performance. The first explores how this learning model can work for its own task. The second is a comparison of results among three types of learning algorithm. The third explores how multiple types of sensory input can be used to learn. This achieves the goal showing HSAT can reinforce the stimuli based on reward information from an environment in real time. Soltoggio and Stanley investigated how to build an ANN learning model based on Hebbian plasticity and rewards [53]. They suggested a modulatory rule, named *reconfigure-and-saturate modulated Hebbian plasticity*, which utilizes neural noise and synaptic weight saturation in order to overcome the limitations of the basic Hebbian learning. The connection between local synaptic plasticity and behavior learning is modulated by two neurotransmitters. They adapted GABAergic neuromodulation for inhibiting selective neural weights and glutamatergic neuromodulation for increasing the

strength of the weights. Their work is actually the basis of the reward-based Hyperbolic Hebbian plasticity in Chapter 3.3 and it is organized in a generalized way in this dissertation. Suh and Hougen designed and constructed *Context-based Adaptive Robot Behavior-Learning Model* (CARB-LM) which is conceptually inspired by Hebbian, anti-Hebbian learning, and selective weight update in neural networks [56]. CARB-LM has two types of learning process: (1) context-based learning and (2) reward-based learning. The former uses past accumulated positive experiences as analogies to current conditions, allowing the robot to infer likely rewarding behaviors, and the latter exploits current reward information so the robot can refine its behaviors based on current experience. They showed its performance by simulating the open environment using ROS and a Gazebo, TurtleBot, where the robot showed substantial learning and greatly outperformed both a hand-coded controller and a randomly wandering robot.

On the other hand, Spike Timing Dependent Plasticity (STDP) refers to the synaptic plasticity which is based on the potentiated timing difference between presynaptic and postsynaptic neuron [61]. STDP uses the timing difference between presynaptic and postsynaptic action potentials. The basic idea of STDP is to strengthen the synaptic weight if presynaptic spike occurs first then postsynaptic spike does later with slight timing difference and is to weaken the synaptic weight if the postsynaptic spike occurs after presynaptic spike. Similar to reward-based Hebbian plasticity, if STDP is learned based on reward information, it is called reward-modulated STDP. However, the reward signal is not just given from an environment but generated by the spike timing difference between actual output and target output spike. This means a given learning model can have maximum reward value when the actual and target output spike

occurs almost simultaneously and will have no reward when they are activated in different timing. van Rossum et al. suggested a stable STDP learning rule and showed that the synaptic weights can evolve with changes of input data [62]. It is also inspired by Hebbian learning; however the difference is considering timing difference between a pre-synapse event and a post-synapse event. For example, if a synaptic event arises before a post-synapse event, the link between them is potentiated; and if the former event occurs after the latter event, the relationship of them is depressed. To demonstrate this model, they showed a stable distribution of the probability of weights. Also, they explained how correlations between input data affect weight changes, especially for potentiation.

4.2 Neurorobotics Learning

There have been vigorous researches on neurorobotics area that typically applies value and reward based neuromodulatory learning into an adaptive and autonomous robot system [23]. Many researchers investigated the role of neuromodulation of a mammal brain and designed their own neural models inspired by the biologically revealed relationship between several neuromodulations and related behaviors. The newly designed neural frameworks are tested for deriving the performance on a mobile robot which can gather the sensory information from the environment and navigate in a given area. In this section, the preceding research works regarding to neurorobotics are categorized into two parts: (1) building biologically plausible learning model by adapting most well-known neuromodulations into a robot behavioral control and (2) building a learning model inspired from both supervised and reward-based learning.

First, Fleisher and Edelman built a synthetic neural model which provides a

tool for analyzing a mammal neural system [14], which is biologically feasible. They defined a neural model as a brain-based device (BBD) if it has following features: (1) a behavioral task is controlled by a synthetic neural process, (2) neural dynamics is biologically plausible, and (3) a BBD includes 10^4 to 10^6 simulated neuronal units and 10^6 to 10^9 synapses similar to a vertebrate brain. The BBD applies value systems to generalize given input signals into perceptual categories based on experience without prior knowledge, which is similar to the way of context-awareness in CALM. They implemented neural function of hippocampus, which is highly related to the navigation ability in a rat, and showed that mobile robots (Darwin X and Darwin XI) can find hidden target location in a maze by using experience-dependent plasticity from the value-dependent learning. In their neural model, dopaminergic neuromodulation plays a role as a positive value responding to currently selected action so that it strengthens the synaptic connection from the input and the selected output which is called episodic memory formation. There are typically used neurotransmitters in modulatory robotics control: (1) dopamine, (2) serotonin, (3) acetylcholine, and (4) noradrenalin. Sporns and Alxeander designed sophisticated neural networks for appetitive learning following rewarded stimuli and aversive learning avoiding punishment, based on EPSP and IPSP neuromodulatory connections [55] [54]. They applied dopaminergic neuromodulation by strengthening neural paths to motor neurons for rewarding stimuli such as red objects and by inhibiting the neural paths for aversive stimuli such as facing blue objects. They showed a mobile robot, called 'Monad', can have biologically plausible neural responses according to each appetitive, aversive, and compounded environment. Cox and Krichmar designed a phasic neuromodulatory neural model for robot control which can sharpen neural pathways which is related to a certain environmental event to select an appropriate behavior [8]. In their neural framework, each

different environmental stimuli activates each different neural modulatory system in different area of a brain and then it causes desired behaviors. The neuromodulatory systems and the corresponding area of a vertebrate brain are connected with the following behaviors: (1) dopamine neuromodulation in ventral tegmental area(VTA) with 'find' behavior, (2) serotonergic neuromodulation in raphe nucleus with 'flee' behavior, and (3) cholinergic neuromodulation in basal forebrain with 'exploration' behavior. They trained a mobile robot, CARL-1, to pick one of three behaviors based on given vision sensory information then showed the robot can activate each different type of neurons to cause appropriate behavior in an open environmental space. Also, Krichmar showed a mobile robot, iRobot Create, can select an appropriate behavior by its own designed neuromodulatory system where each neuromodulation corresponds to a possible unexpected event occurred in an open environment [24] [21] [22]. He suggested a neural network for robot action selection which is motivated from the principles of neuromodulatory systems: (1) dopaminergic, (2) serotonergic, (3) cholinergic, and (4) noradrenergic neuromodulation. The neural network exploits the neuromodulations for a robot decision making process by setting input events neurons and output behavioral neurons. The input events are categorized into two groups: stressful and interesting one. The stressful events connect to the serotonergic neuromodulation, which is in charge of controlling risk-taking(harmful) or withdrawn behaviors, and the interesting one is associated with the dopaminergic neuromodulation, which takes charge of altering curiosity-taking(reward-seeking) or exploratory behaviors. Avery et al. built a neural model for an attention behavior by focusing on the correlation between the cholinergic neuromodulation in the basal forebrain and noradrenergic neuromodulation in locus coeruleus area of a vertebrate brain [7]. In an uncertain decision making process, they verified a simulated rodent robot can pay attention

to expected uncertain light events by picking its head direction; and it can be aware of unexpected uncertain light events by not moving in current position based on the combination of cholinergic and noradrenergic neuromodulation. In their experiments, they put more light events for expected uncertain events and less lights for unexpected one. Based on the research works of Krichmar, Prince and Samanta showed how a mobile robot can select an appropriate behavior by context-based neuromodulation [39]. They designed 3-layered neural network where first layer presents 4 possible events as contexts (BUMP, BEAM, OBJECT, BATTERY), second layer takes neuromodulatory controls for robot behaviors, and third layer indicates 4 possible behavioral states (WallFollow, OpenField, ExploreObject, Home). In neuromodulatory layers, they used dopaminergic and serotonergic synaptic connections to risk-taking and risk-aversive behaviors respectively. With this neural network, a robot can make association between a given context and exploratory or exploitative behaviors from the designed neuromodulations by taking one event at a learning time step.

Second, there are preceding resesarch works on building a novel learning model inspired by both supervised and reward-based learning. Uchibe and Doya used reinforcement learning with gradient projection for finding optimized learning parameters [59]. They used the gradient of average reward at a learning step and calculated gradient projection onto current constraints space. This research work is part of the Cyber Rodent Project which shows cyber agents can increase the average of reward by appropriately selecting one of two behaviors: (1) foraging and (2) mating. Noda et al. introduced a novel arbitrary-depth neural network for generalizing robot behaviors based on multimodal temporal sequence integration learning [34]. They applied Hessian-free neural optimization for tuning the learning weights and used time-delay neural

networks for recognizing robot motion and temporal sequence learning. With 12-layered neural network, they showed a robot NAO could generalize 6 kinds of behavior from 3 different type of inputs: joint angular, image, and sound features.

The previously mentioned research works showed the promising experimental results with their newly designed neural models. Seth et al. organized the effects of the neurorobotics researches as follows: (1) a chance of understanding of a mammal brain's behavior selection, (2) an opportunity of comparing the implemented empirical data and a mammal's data, (3) providing a testing framework for discovered brain theories, and (4) giving a foundation for a better robotics [3]. Therefore, the neurorobotics approach aims to provide more flexible, efficient, and autonomous robot controllers. However, most of the novel neural models are based on simulating or mimicking the specific functions of neuromodulations or neural dynamics; and they are tested on their own target specific domains. Based on the above research works, CALM aims to introduce a robot brain by endowing with a generalized arbitrary-depth neural optimization process based on reward and experience-based knowledge base which serves as a memory. Moreover, CALM-nepLRB adapts specific neurobiological features of several natural animal behaviors (bat, moth, ell, honeybee, crayfish, and drosophila) as well as the well-know features of neuromodulation (serotonin and dopamine). Especially, in designing dopaminergic neurons, more detail role of dopamine with two different type of dopaminergic receptors are investigated, which is described in Chapter 5.5. With those unique features, the outperforming performance of CALM-nepLRB is described in Chapter 6.

Note that CALM is not for generating a certain animal's neural dynamics nor for showing that a computational model serves as the same functions of

a certain animal. It is rather for introducing a novel way of building a robot brain and for showing its flexible features and promising results which provides a generalized neural framework.

4.3 Context-based Robot Learning

In the area of cognitive robotics, there are memory-related research works for an adaptive and autonomous learning which is considered as necessary part of a robot brain. The memory-related investigations can be classified into 3 approaches: (1) animal behavioral learning, (2) hippocampal learning, and (3) knowledge-based learning. First, animal behavioral learning is embodied based on instrumental learning or conditioning learning in animal behaviors [45]. In neurobiological approach, the memory in a learning model is designed and implemented based on an hippocampus neurophysiological structure of a vertebrate brain, especially from a rat [14]. In knowledge-based approach, a memory has certain type of structure such as a frame or ontology so that a main controller can infer an appropriate behavior [52] [17] [46] [32].

First, Saksida et al introduced a computational robot behavior shaping learning model for generating an appropriate behavior [45] based in instrumental learning. This model generates a target behavior based on behavior editing technique, which can produce a new behavior, modify an existing behavior, or exterminate a behavior from the pre-existing ones (originally hard-wired behaviors). As a results of learning, this approach gives a behavioral topology with behavior sequences which is a computational model of animal behavior in a certain domain.

Second, some research works are inspired by a vertebrate memory learning

process based on hippocampus. Thomas et al. applied the probabilistic learning model, called HyGene, to generalize hypotheses from a memory which is pack of all possible events from a world [58]. The HyGene mimics human judgement process and includes three types of memory component: (1) working memory for saving possible hypotheses, (2) episodic memory for comparing which hypothesis has high probability given an event data, (3) semantic memory for saving generalized hypotheses. They showed this probabilistic learning model can theoretically produce the semantic memory structure and thus can make judgment through 3 types of simulation. Salado et al. designed and implemented a high-level robot memory architecture, called evolutionary-based MDB(Multilevel Darwinist Brain), including STM(Short-Term Memory) and LTM(Long-Term Memory) based on context detection in an dynamic environment [46]. MDB optimizes its neural networks based on the model errors which are calculated from the differences between current and pre-defined satisfactory world model and then it saves the successful model status in to LTM per one context. In this case, if the context is changed in an dynamic environment in future learning step, MDB performs model recovery for stable LTM. By having this LTM, a robot with AIBO model could reduce its number of iterations in performing desired behavior when repeated changed contexts occurred; this is because the LTM stores a successful learning model per one context from the previous experiences. In this regard, the basic idea of storing the past successful learning model status is similar to EKB of CALM but CALM; thus the EKB can be also considered as LTM which includes contextual information including the neural connection information. For optimizing the neural network, it is also notable that MDB used a Differential Evolution algorithm to adjust the learning parameters while CALM exploits reward- and experience-based logistic regression for generalized arbitrary-depth neural networks which is not based on pre-defined satisfactory

world model.

Third, in the knowledge-based learning, there are vital research works on building a robot controller based on ontology or contextual information. on building an robot memory by separating long-term and short-term memory. Suh et al. designed and implemented memory of a mobile service robot for its behavioral controller, which is called OMRKF(Ontology-based Multi-layered Robot Knowledge Framework) [12]. OMRKF includes 4-layered knowledges from low-level knowledge such as input sensory information and high-level one such as behavioral tasks and the final behavior is inferred from the lowest layer with each inference rules based on logic-based language. They showed a mobile robot with OMRKF can perform a cup delivery service in a real environment. Furthermore, Lim et al. also introduced a weighted Action-coupled Semantic Network(wASN) based on the OMRKF which supports a robot select an appropriate action through rule-based neural node selection [26] [27] [28]. They showed The detailed process of inferencing the robot behavior from the ontology-based knowledge base and showed how a robot completed its high-level service only from a well-designed memory structure.

Moore and Pham proposed a learning model for Tunnel Boring Machine(TBM) performance to predict contexts including machine performance and disaster risk in tunneling project [32]. They used contextual information, knowledgebase, and Hybrid Artificial Neural Networks to find most appropriate parameters of TBM performance based on a feedforward artificial neural network as a supervised learning and a fuzzy reasoning evaluation with SOM as an unsupervised learning. In evaluating Root Mean Square Error(RMSE), they showed this context-based approach is better than existing statistical model in terms of PR(Penetration Rate) prediction. Rocket et al. also introduced experience-based robot learning

model which is a part of the project RACE(Robustness by Autonomous Competence Enhancement) [42]. They build an ontology-based robot knowledge base which supports a robot behavioral inference mechanism through OWL-based domain-specific rules. The experiences in this model are formed by logical expression such as the sequential events; and an similar experience is extracted by the inference rules while CALM retrieves the experience-based on most similar context without inferences. Saeedi et al. introduced context-aware brain-computer interface for developing shared control system which makes decision from the cognitive process based on internal contexts and from the manual control of the user [44].

The previously mentioned research works in cognitive robotics show how a robot can be successfully controlled by inferring or shaping its own type of memory formation. However, the evaluations of the memory formation are based on a domain-specific knowledge and most importantly most of the robots started its learning process with the prior-knowledge of the world, which is not bootstrapping. In this regard, CALM shows how a learning model shapes its own experience-based knowledge based(EKB), which serves as a hippocampus in a vertebrate animal, from the scratch and evaluated its performance based on several generalized synthetic data sets.

Chapter 5

The Context-Aware Learning Model (CALM)

As briefly introduced in Chapter 1, CALM includes four different types of algorithms, each of which has different learning principles and effects. In this chapter, the overall CALM architecture is described, then each algorithm is covered in its own section.

5.1 System Architecture

CALM is a hybrid learning model since it does not belong to one specific classical learning model; it takes advantage of different learning approaches. However, it is not limited to having combinations of good features from different existing machine learning methodologies, it is a novel learning model utilizing existing well demonstrated features and new learning features.

Also, CALM is intended to be used in various domains including but not limited to robotics. The system structure and computational learning process are generalized thus all the algorithms of CALM can be applied to domains with different input sizes, output sizes, neural network arbitrary depths, and learning parameters: learning rate, regularization rate, number of iterations, similarity rate, and experience-power rate. Before getting into the details of CALM, Table 5.1 shows the symbols that are used in CALM and their corresponding meanings.

Index	Symbol	Meaning
1	T	Maximum discrete learning step
2	t	Discrete learning step, $t \in [1, T]$
3	n	Total number of context features/attributes
4	j	Context feature (input node) indicator, $j \in [1, n]$
5	m	Total number of experiences in EKB (size of EKB)
6	i	Experience indicator in EKB, $i \in [1, m]$
7	ITR	Total number of iterations in a learning step
8	L	Total number of layers of an ANN including input layer
9	l	Layer indicator, $l \in [1, L]$
10	K_l	Total number of nodes at l^{th} layer, $K_1 = n$
11	k	Node indicator in each layer, $k \in [1, K_l]$
12	$X(t)$	Context vector at t^{th} learning step
13	$x_j(t)$	j^{th} context feature at t^{th} learning step
14	$\theta^{(l)}$	Weight vector or matrix on l^{th} layer of a CALM-ANN
15	$z_k^{(l)}(t)$	k^{th} net value on l^{th} layer at t^{th} learning step
16	$a_k^{(l)}(t)$	k^{th} actual output value on l^{th} layer at t^{th} learning step
17	$\delta_k^{(l)}(t)$	k^{th} error value on l^{th} layer at t^{th} learning step
18	$son(t)$	Selected actual output node at t^{th} learning step
19	η	Learning rate
20	λ	Regularization rate
22	ϵ	Similarity rate
22	γ	Experience power rate
23	$J(t)$	The value of cost function at t^{th} learning step
24	$r(t)$	Reward value at t^{th} learning step, $r(t) \in \{0, 1\}$
25	$R(t)$	Quasi-target output vector at t^{th} learning step based on $r(t)$
26	$R_k(t)$	k^{th} quasi-target output value at t^{th} learning step based on $r(t)$
27	$LRBX$	Learning input for arbitrary-depth optimization
28	$LRBY$	learning output for arbitrary-depth optimization
29	IHV	Inhibition value

Table 5.1: CALM Symbols

CALM consists of 10 components: (1) Sensory System, (2) Context Supplier, (3) CALM-ANN, (4) Motor System (Actuator), (5) Observer, (6) Reward Policy Storage, (7) CALM-Learner, (8) Learning Rule Storage, (9) EKB, and (10) Experience-based Knowledge Base (EKB) Manager. Figure 5.1 gives the overall system structural flow and shows how the components are intertwined in CALM.

Note that Figure 5.1 represents both essential and optional components. The EKB and EKB Manager are optional components since they are not used in CALM-rLRB; those components and the corresponding path flows are denoted with dashed lines, rather than solid lines.

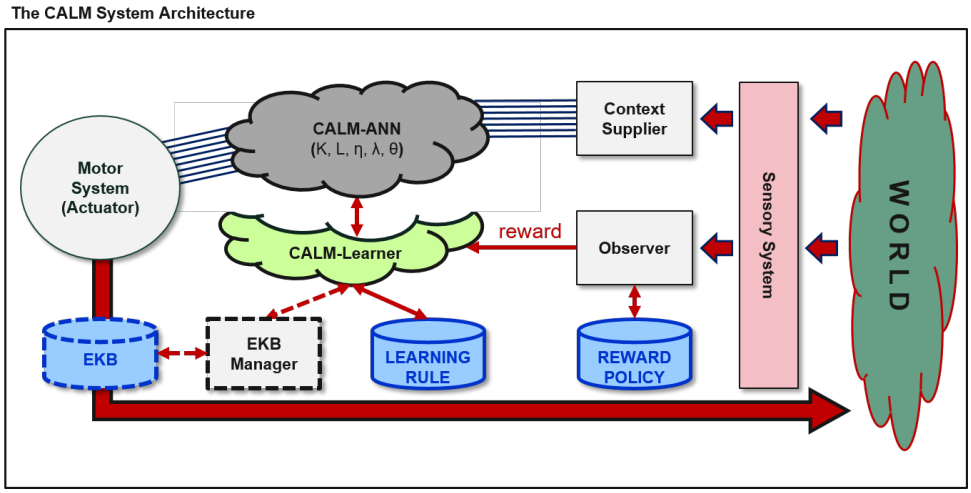


Figure 5.1: CALM System Architecture

Sensory System The *Sensory System* is in charge of gathering input data directly from sensory device(s) such as lasers, image sensors, etc. CALM is intended for use in non-robotics applications as well as in robotics applications so it is not necessarily the case that input will always be sensory data. The Sensory System is symbolically named in that it takes raw data directly from other objects without internal processing. In non-robotics applications, for example, the input data could be also a binary file or a user-defined file format as needed.

Context Supplier The *Context Supplier* is devised for efficient data handling where the data is given from the Sensory System. It is in charge of processing the given data before feeding it into input nodes of the CALM-ANN. Since the data can be from one or various input types, possible data processes can be feature

scaling, input encoding, and/or data compression. As a simple example, if it is assumed that a raw data transaction from a laser sensor at one time is a binary array with the length of 500, then it can be compressed into an array with the length of 10 by taking the average value each 50 element values sequentially; or, it can be compressed by using principle component analysis. Input encoding or feature scaling helps reduce computational complexity and leads to more efficient learning [19] and thus the Context Supplier aims to support multi-modal input system by generating more appropriate inputs for the CALM-ANN based on the low-level data from the Sensory System. Each vector of processed data from the Context Supplier is called a *context*.

CALM-ANN The *CALM-ANN* is an artificial neural network in CALM that interacts with the Context Supplier, CALM-Learner, and Motor System. There are two types of CALM-ANN; any given CALM system will have an ANN of one type or the other. The first type is a generalized, arbitrary-depth, neural network as shown in Figure 3.17 in Section 3.2, which is used for CALM-rLRB, CALM-eLRB, and CALM-epLRB algorithms. The second type of CALM-ANN is called *CALM-nepLRB-ANN* which is a novel, bio-inspired variant of the neural network designed specifically for use with the CALM-nepLRB algorithm. For either type of ANN, the CALM-ANN takes input signals as a context vector from Context Supplier, propagates the signals forward from the input layer to the output layer of the CALM-ANN, and selects the output node with the highest activation value, which is considered to be the most appropriate output at the current learning step. Note that CALM-ANN is a generalized neural network and thus a user can select the depth of the ANN, the number of input nodes, and/or the number of output nodes appropriate.

Motor System (Actuator) The *Motor System* is in charge of taking the selected output from the CALM-ANN and carrying out the corresponding action in the environment in which the agent is found. The actions can be either virtual or physical depending on the domain. In robotics, CALM considers that each action or behavior corresponds to one output node in the CALM-ANN. For example, each output node in the ANN may refer to a different robot behavior and if one of them is selected and sent to the Motor System, the robot carries out the corresponding behavior. In this case, it is expected that executing the behavior will affect the environment in either a positive or negative way. On the other hand, in non-robotics applications, we can consider that each action is virtual and thus the Motor System does not cause any physical movement but lets the environment know which neural output is selected. For example, if CALM is used to recognize human faces, the Motor System will announce which output is selected to get a feedback value from the environment. In this regard, the Motor System, like the Sensory System, is also named symbolically so as to provide CALM with the flexibility to be applied in various domains.

Observer The *Observer* acquires feedback from the environment after the action is carried out by the Motor System. The Observer checks the effect of selecting an output node based on the reward policy saved in Reward Policy Storage. The way of checking the effect is based on comparing the environmental status before and after selecting the output node; environmental changes are compared with criteria from Reward Policy Storage. If it decides the selected output node causes positive effects in the environment, it gives out reward; if it recognizes negative effects, it sends a punishment signal instead of reward. In this way, the Observer plays an important role in deciding whether the performed behavior from the Motor System is good or bad through interactions with the environment.

Note that it is also possible for the Observer to get feedback by directly interacting with the other sources, such as a trainer. For example, if an output node is selected in CALM-ANN, then the Motor System sends the information to a trainer who can give the Observer the feedback value directly.

Reward Policy Storage *Reward Policy Storage* holds the expected effects of selecting each output node or of executing each behavior. For a simple example in robotics, consider a case which each behavior has its own goal in affecting the environment. If a robot executes a behavior called 'GO_FORWARD', it is expected that the position of the robot is changed after going forward. In this case, reward policy rule for 'GO_FORWARD' would be: if the position is changed after executing 'GO_FORWARD', the effect is positive; or if the position is not changed at all, the effect is negative. In this way, Reward Policy Storage provides the Observer with appropriate criteria of checking how a robot is doing so that Observer can decide whether a robot gets a reward or not. Note that the reward policy depends on the user's design and a target domain.

CALM-Learner The *CALM-Learner* embraces all the CALM algorithms: CALM-rLRB, CALM-eLRB, CALM-epLRB, and CALM-nepLRB; each algorithm exploits its own CALM-Learner. Basically, the CALM-Learner adjusts the CALM-ANN based on weight update rule(s) found in Learning Rule Storage. Also, it utilizes the EKB if its algorithm utilizes experiences. Each CALM algorithm is described in detail in its own section.

Learning Rule Storage *Learning Rule Storage* has all of the weight update rules which may be used by the CALM-Learner. The reason for separating the CALM-Learner and Learning Rule Storage is to give flexibility in designing

CALM learning systems so that the CALM-Learner can select learning rule(s) based on its algorithm. For example, if a new algorithm and new learning rule are added to CALM and it utilizes two learning rules, where the one is an existing weight update rule and the other is a newly added one, then it can select the existing learning rule which is also used in another CALM-Learner and the newly added learning rule from Learning Rule Storage, when each is appropriate.

Note that CALM-rLRB and CALM-eLRB both use Logistic Regression Backpropagation (LRB) based on gradient descent optimization. The difference between the way LRB is used in CALM-rLRB and in CALM-eLRB is based on the EKB; CALM-rLRB only uses the current context while CALM-eLRB also uses stored past rewarding experiences from the EKB for updating weights. CALM-epLRB exploits two weight update rules where the one is LRB and the other is a novel weight update rule called Selective-Power-Update (SPU). Therefore, the CALM-epLRB learning rule encompasses CALM-eLRB and SPU. On the other hand, the CALM-nepLRB learning rule encompasses CALM-rLRB, CALM-eLRB, and CALM-epLRB by incorporating additional neurobiological features into them. It is notable that only CALM-nepLRB is able to choose if it will use the EKB or only use the current context like CALM-rLRB. More specifically, the CALM-nepLRB Learner selects one of the learning rules between CALM-rLRB, CALM-eLRB, and CALM-epLRB with the additional features, which gives it flexibility in choosing the most appropriate algorithm based on the current learning status. Each algorithm is described in detail in its own section.

Experience-based Knowledge Base (EKB) The *EKB* stores past positive experiences. The EKB serves the same functions as the Empirical Context-based

Knowledge Base (ECKB) in our previous research [56]. Positive experience means that a selected output in a certain context triggered positive effects and thus received reward. This is saved into the EKB before a new context is given so that it can be used by the CALM-Learner. Note that CALM-rLRB is not experience-based, thus it does not use the EKB; the other three algorithms do use the EKB. The details of the EKB are described in Section 5.3.

EKB Manager The *EKB Manager* is an interface to the EKB which carries out three major tasks: (1) storing the current experience into the EKB if it is rewarding, (2) retrieving necessary information from EKB, and (3) performing knowledge optimization. In order to retrieve appropriate information from the EKB, the EKB Manager searches in order to find an experience in the EKB and then transfers the search results to the CALM-Learner. The search criteria is based on the Euclidean distance between a stored context. To be selected, the distance must be less than the similarity rate ϵ. Note that in this case the current context is not given from the Context Supplier but given from the CALM-Learner the CALM-Learner already possesses the current context vector from the CALM-ANN and the Context Supplier.

Knowledge optimization can be considered as a knowledge base mechanism that aims to increase memory efficiency by reducing knowledge redundancy and correcting knowledge inconsistency. Knowledge redundancy occurs when there are similar experiences that are almost alike. This redundancy can be avoided by making one composite experience which represents all the most similar experiences. Knowledge inconsistency happens when the positive experiences stored in the past are no longer reliable due to changed environment. In other words, inconsistency refers to the case when past experiences are no longer accurate

reflections of how to get positive reward values in the changed environment. In this case, the EKB Manager should resolve this issue by removing old experiences and updating the knowledge base with more recent experiences. In this way, the EKB Manager supports time-sensitive learning through knowledge optimization.

5.2 CALM-rLRB

In this section, the concept of CALM-rLRB is explained including its motivation, the detailed algorithm, its learning effects, and its fundamental mathematical principles. Of particular importance is the method by which reward-based logistic regression neural optimization is performed as it is also used in CALM-eLRB, CALM-epLRB, and CALM-nepLRB.

5.2.1 CALM-rLRB Features

CALM-rLRB performs reward-based learning. In reward-based learning, there is no *target output* (vector of desired responses) for a given input. Instead, it supports interactive online learning by obtaining evaluative feedback from the environment and applying that to its learning process. In contrast, supervised neural learning with logistic regression backpropagation (LRB) or least mean squares (LMS or delta rule) has a target output for each input and uses them to optimize its neural weights by reducing the errors between the computed actual outputs from the system and the provided target output. In this case, it is important to point out that supervised learning usually shows its advantages in offline batch learning with appropriate amounts of given training data including target output. Therefore it can be considered that (1) supervised learning is not designed to be used for dynamic adaptive learning in unknown

environments, but it is beneficial for optimization based on given target outputs; (2) reward-based learning is not devised for optimization but it is a powerful way to explore uncertain environments interactively. Based on this, the central question of CALM-rLRB is how to optimize an arbitrary-depth neural network without target outputs and how to meld reward information appropriately into the optimization process.

The basic idea of CALM-rLRB is to generate "quasi-target output" based on reward and exploit the generated quasi-target output in optimizing its neural weights, instead of depending on a teacher saying which output is correct for a given input. In this dissertation, quasi-target output is newly defined as follows.

Quasi-target output is similar to target output in that both of them are used in the optimization; the learning equations are intended to adjust the weights to produce the desired outputs. However, there are major distinctions between them. First, in general supervised learning, target output is given from outside the learning process as labeled training data (that is, the "right answer" for each input vector). In CALM learning, quasi-target output is inferred by the learning model during the learning process. Second, in general supervised learning, target output can have any combination of values while quasi-target output in CALM has restrictions based on the inference method used. The method for inferring quasi-target output is described with each CALM algorithm.

General supervised ANN learning has the following step. (1) Take input, (2) do forward propagation, (3) select the output node that has the maximum output value, (4) calculate errors between actual and target output, and (5) optimize neural network weights depending on the cost function (LMS, LRB, etc.) and weight update types (GD, CGD, etc.). Note that, in this research,

optimization involves two processes: (1) calculating the cost function based on the errors and (2) performing weight updates.

In contrast, reward-based Hebbian learning has the following step. (1) Take input, (2) do forward propagation, (3) select the output node that has the maximum output value, (4) perform the behavioral task corresponding to the selected output node, (5) get feedback from the environment through an interaction process (e.g., sensing the world or getting feedback from a trainer), (6) perform weight updates based on reward information. Note that often it is assumed that there are two values of feedback: the first is positive and the second is negative. In CALM, the feedback value can be either 0 or 1; a reward of 0 is used for negative feedback (incorrect behavior) and a reward of 1 is used for positive feedback (correct behavior).

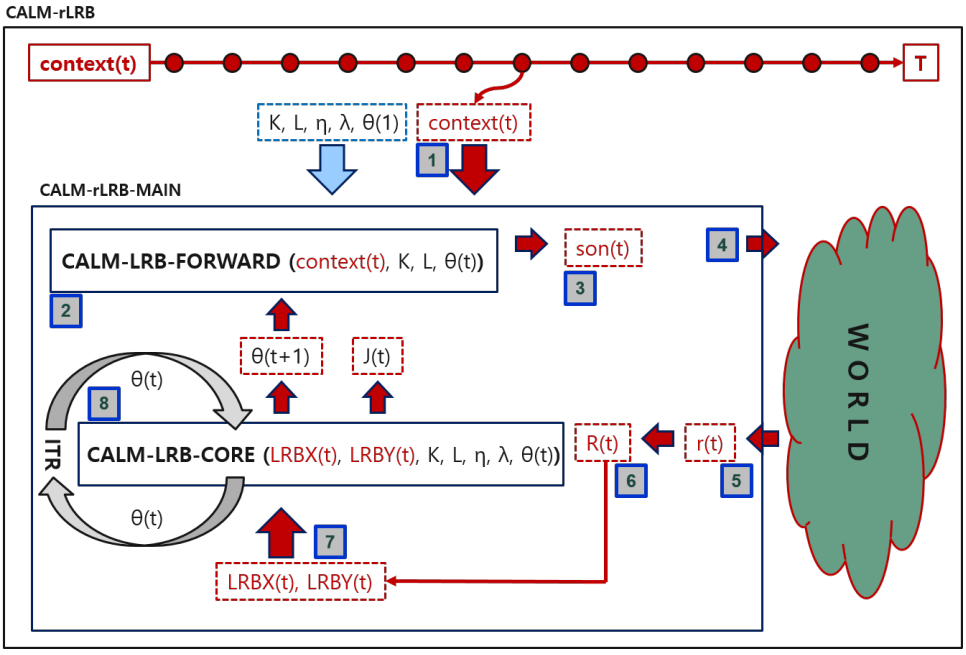


Figure 5.2: CALM-rLRB Algorithm Diagram

CALM-rLRB has more steps since it generates quasi-target output based on reward and utilizes it in its optimization. Figure 5.2 shows how CALM-rLRB performs reward-based neural learning. The steps of CALM-rLRB are as follows. (1) Take contextual input $context(t)$, (2) do forward propagation, (3) select the output node that has the maximum output value; this is denoted as the selected output node $son(t)$, (4) perform the behavioral task corresponding to the selected output node, (5) get the reward value $r(t)$ from the environment, (6) generate quasi-target output $R(t)$ based on reward value $r(t)$, (7) set learning input $LRBX(t)$ and learning output $LRBY(t)$ for the logistic regression optimization which performs logistic regression optimization with gradient descent by taking $LRBX(t)$, $LRBY(t)$, and the current neural weights $\theta(t)$ as inputs. In this last step, the output is the updated neural weights $\theta(t+1)$ and the cost function value $J(t)$, which are calculated at the end of the certain number of iterations ITR in the CALM-LRB-CORE optimization process. $\theta(t+1)$ is used in the next learning step with the new context $context(t+1)$ and $J(t)$ is used for only evaluation (and thus looks like dead end in Figure 5.2); cost function values are analyzed in Section 6.2.2. The detailed computational process of these steps are explained in Section 5.2.3.

In CALM, input is considered as a low-level data while context refers to high-level data. It is also possible that input data and context have exactly same values in the same form; but ultimately context implies contextual information which can be integrated from several types of input with/without input encoding, feature scaling, and/or data compression.

$LRBX(t)$ and $LRBY(t)$ are the names for the input vector (or matrix) and desired output vector (or matrix), respectively, used for logistic regression optimization. These are generated by CALM-rLRB based on current context

context(t) and quasi-target output $R(t)$ during the learning process; in other words, it is the $LRBX(t)$ and $LRBY(t)$ that are generated by CALM-Learner and are fed into logistic regression optimization process whereas *context*(t) is.

In CALM-rLRB, the quasi-target output $R(t)$ is generated in one of two ways depending on the reward value. First, if the reward value is 1 ($r(t) = 1$), this means the selected output node at learning step t is good choice for the current context, *context*(t). In this case, the quasi-target output value, which corresponds to the selected output node, is set to be 1 ($R_{son(t)}(t) \leftarrow 1$); and the quasi-target outputs corresponding to the other nodes are set to 0 ($R_k(t) \leftarrow 0$ where $k \neq son(t)$). In this way, the quasi-target output implies that the selected output node is expected to be the right answer if the agent faces to a similar context in a future learning step and, moreover, that none of the other outputs are expected to be the right answer in that context.

Second, if the reward value is 0 ($r(t) = 0$), the quasi-target output value corresponding to the selected actual output is set to be 0 ($R_{son(t)}(t) \leftarrow 0$) while the other quasi-target values are set to 1 ($R_k(t) \leftarrow 1$ where $k \neq son(t)$). In this way, the quasi-target output implies that the currently selected output node is a bad choice for the current context so it should have its weights adjusted so that it becomes less likely to be selected in a similar context in the future, and, moreover, that one of the other output nodes is likely to be the correct choice so the other output nodes will have increased opportunities to be selected in the future by setting their target values to 1.

After setting the quasi-target output $R(t)$ based on the reward value $r(t)$ as above, CALM-rLRB is ready to generate learning input $LRBX(t)$ and learning output $LRBY(t)$ to optimize CALM-ANN with logistic regression backpropaga-

tion. Note that each algorithm in CALM has its own way of generating $LRBX(t)$ and $LRBY(t)$. In CALM-rLRB, $LRBX(t)$ is simply set from $context(t)$ and $LRBY(t)$ is set to be $R(t)$. With this setting, CALM-rLRB learns by trial and error which output node should be selected in each context based on the reward information.

Step	$context(t)$	$son(t)$	$r(t)$	$R(t)$	$LRBX(t)$	$LRBY(t)$	$sw(t)$
t	$[1,1,1]$	3	1	$[0,0,1]$	$[1,1,1]$	$[0,0,1]$	θ_3
$t+1$	$[1,2,2]$	3	0	$[1,1,0]$	$[1,2,2]$	$[1,1,0]$	θ_3
$t+2$	$[1,3,3]$	2	1	$[0,1,0]$	$[1,3,3]$	$[0,1,0]$	θ_2
$t+3$	$[1,4,4]$	2	1	$[0,1,0]$	$[1,3,3]$	$[0,1,0]$	θ_2

Table 5.2: CALM-rLRB Learning Example

To clarify the learning process, this section introduces a simple learning example as shown in Table 5.2. The learning example shows the necessary information for learning: (1) the contexts given, (2) the outputs selected, and (3) the feedbacks received for each selected output. Also, the example includes the following assumptions: (1) CALM-ANN is a 2-layered neural network, (2) the context input dimension is three including a bias value, (3) and the number of possible outputs is three.

This simple example does not include all elements of the learning process but does explain how to set the quasi-target output $R(t)$ based on reward and how to set the learning input $LRBX(t)$ and learning output $LRBY(t)$ for optimization. Note that $sw(t)$ refers to $selected\_weight\_vector(t)$, which is the weight vector corresponding to the selected actual output node $son(t)$ at learning step t.

In the first row of the table, when the learning step is t and the context is $[1,1,1]$, the selected output node is 3 and it is assumed that the selected

output node acquires a reward value of 1 from the environment. Given that $r(t) = 1$ and $son(t) = 3$, the quasi-target output $R(t)$ generated is $[0, 0, 1]$. At this point, CALM is ready to set $LRBX(t)$ and $LRBY(t)$, which will be feed to the logistic regression optimization. In CALM-rLRB, $LRBX(t)$ is simply set equal to $context(t)$ and $LRBY(t)$ is set equal to $R(t)$. The expectation of this approach is that this will reduce errors between the current actual output and the generated quasi-target output for the current context. Considering the role of weight updates as given in Section 3.2, with this approach, CALM-rLRB is expected to have two effects in the logistic-regression learning process. The first effect is that θ_3 will be updated based on the quasi-target output value of 1 for the output node 3 and thus it will increases the value of that output node in the feedforward process in subsequent learning steps if the same (or a similar) context is given. the second effect is that the other neural weights θ_1 and θ_2 will be updated based on the quasi-target outputs of 0 and and thus the values of these non-selected actual output nodes will be reduced in subsequent learning steps if same (or a similar) context is given.

However, in row two of the table, when the learning step is $t + 1$ and the context is $[1, 2, 2]$, the reward is 0 and $son(t)$ is 3. In this case, the quasi-target output $R(t)$ is set to be $[1, 1, 0]$, and $LRBX(t)$ and $LRBY(t)$ are set to be the new context and the new quasi-target output, as shown in the table. This also has two learning effects on the neural weights. First, it will cause θ_3 to be updated such that it will give decreased actual output value for output node 3 at subsequent learning steps if the same (or a similar) situation occurs. Second, θ_1 and θ_2 will be adjusted so as to have greater opportunities to be selected at subsequent learning steps if the same (or a similar) context occurs.

In this way, CALM-rLRB provides reward-based neural context-awareness,

which incorporates advantages from both supervised and reward-based learning; this is the basic principle of CALM. The mathematical processes are explained in Section 5.2.3, 5.2.4, and 5.2.5.

5.2.2 CALM-rLRB-ANN

The neural network for CALM-rLRB is same as the generalized, arbitrary-depth, ANN shown in Figure 3.17 in Section 3.2.

5.2.3 CALM-rLRB Learning

In this section, we will see how CALM-rLRB performs reward-based learning iteratively. In CALM-rLRB, there are two primary functions: (1) CALM-rLRB-MAIN and (2) CALM-LRB-CORE. Algorithm 10 shows the main flow of CALM-rLRB and Algorithm 11 shows how to perform logistic regression optimization with the learning input and learning output which are generated based on the current context and reward. In other words, CALM-LRB-CORE refers to the logistic regression optimization algorithm in CALM which is based on a generalized, arbitrary-depth, ANN as shown in Algorithm 6.

Algorithm 10 CALM-rLRB-MAIN Pseudocode

Given T, ITR, n, L, K_1, $\cdots$, K_l, $\cdots$, K_L, η, λ, ϵ, γ
for $l = 1$ **to** $L - 1$ **do**
  Init $\theta^{(l)} \in \mathbb{R}^{K_{l+1} \times (K_l+1)}$, $\theta^{(l)}(1) \leftarrow \theta^{(l)}$
end for
for $t = 1$ **to** T **do**
  (1) CONTEXT ACQUISITION
  $context(t) \in \mathbb{R}^{n \times 1}$
  Add bias for a context $\rightarrow$ $context(t) \in \mathbb{R}^{(n+1) \times 1}$
  (2) FORWARD PROPAGATION
  $a^{(1)}(t) \leftarrow context(t)$
  for $l = 2$ **to** L **do**
    $z^{(l)}(t) \leftarrow \theta^{(l-1)}(t) a^{(l-1)}(t)$
    $a^{(l)}(t) \leftarrow f(z^{(l)}(t))$
    Add $a_0^{(l)}(t) \leftarrow +1$ for bias at each layer.
  end for
  (3) OUTPUT SELECTION
  $son(t) \leftarrow max_k\{a_k^{(L)}\}$
  (4) BEHAVIORAL TASK
  (5) REWARD ACQUISITION
  Get $r(t) \in \{0, 1\}$ from environment
  (6) SET UP $R(t) \in \mathbb{R}^{K \times 1}$ based on $r(t)$
  if $r(t) = 1$ **then**
    $R_k(t) \leftarrow 1$, if $k = son(t)$
    $R_k(t) \leftarrow 0$, if $k \neq son(t)$
  else if $r(t) = 0$ **then**
    $R_k(t) \leftarrow 0$, if $k = son(t)$
    $R_k(t) \leftarrow 1$, if $k \neq son(t)$
  end if
  (7) SET UP $LRBX(t)$ AND $LRBY(t)$
  $LRBX(t) \leftarrow context(t)^\top \in \mathbb{R}^{1 \times (n+1)}$
  $LRBY(t) \leftarrow R(t)^\top \in \mathbb{R}^{1 \times K}$
  (8) CALL CALM-LRB-CORE OPTIMIZATION
  $[J(t), \theta(t+1)] \leftarrow$ CALM-LRB-CORE ($LRBX(t)$, $LRBY(t)$, $\theta(t)$)
end for

In CALM-rLRB-MAIN, it first sets the learning parameters and initializes the neural weights for each layer. After the initialization, it performs the eight steps iteratively as shown in the Figure 5.2 at each learning step t: (1) Take contextual input $context(t)$, (2) do forward propagation, (3) select maximum

output node $son(t)$ at the last layer, (4) perform the behavioral task corresponding to the selected output node, (5) get reward $r(t)$, (6) generate the quasi-target output $R(t)$ based on the reward, (7) set the $LRBX(t)$ and $LRBY(t)$ based on the $context(t)$ and $R(t)$, respectively, then (8) optimize by calling the function CALM-LRB-CORE, which performs logistic regression optimization by taking $LRBX(t)$, $LRBY(t)$, and the current neural weights $\theta(t)$ as inputs. The output of LRB-CORE is the value of the cost function $J(t)$ and the newly updated neural weights $\theta(t+1)$.

Note that CALM-LRB-CORE also takes all the learning parameters: T, ITR, n, L, K_1, $\cdots$, K_l, $\cdots$, K_L, η, λ, ϵ, γ as well as $LRBX(t)$, $LRBY(t)$, and $\theta(t)$; for simplicity, all parameters are omitted on the pseudocode Algorithm 10.

• The context input vector $context(t)$ represents all n CALM-ANN input values and the bias node at learning step t. Note that $context(t)$ is the same as $a^{(1)}(t)$ for the consistency of the computational notation as explained in Section 3.2.

$$context(t) = \begin{bmatrix} x_0(t) \\ x_1(t) \\ \vdots \\ x_j(t) \\ \vdots \\ x_n(t) \end{bmatrix}_{(n+1)\times 1} ; a^{(1)}(t) = \begin{bmatrix} a_0^{(1)}(t) \\ a_1^{(1)}(t) \\ \vdots \\ a_j^{(1)}(t) \\ \vdots \\ a_{K_1}^{(1)}(t) \end{bmatrix}_{(K_1+1)\times 1} = context(t)$$

where $x_0(t) = a_0(t) = 1$ for the bias.

• Weight matrix $\theta^{(l)}(t)$ represents all weights on the l^{th} layer at learning step t.

Note that $\theta(t)$ refers to all weight matrices over all layers.

$$\theta^{(l)}(t) = \begin{bmatrix} \theta_1^{(l)\top}(t) \\ \theta_2^{(l)\top}(t) \\ \vdots \\ \theta_k^{(l)\top}(t) \\ \vdots \\ \theta_{K_{l+1}}^{(l)\top}(t) \end{bmatrix}_{K_{l+1}\times(K_l+1)} ; \ \theta_k^{(l)}(t) = \begin{bmatrix} \theta_{k0}^{(l)}(t) \\ \theta_{k1}^{(l)}(t) \\ \vdots \\ \theta_{kj}^{(l)}(t) \\ \vdots \\ \theta_{kK_l}^{(l)}(t) \end{bmatrix}_{(K_l+1)\times 1} ; \ \bar{\theta_k}^{(l)}(t) = \begin{bmatrix} 0 \\ \theta_{k1}^{(l)}(t) \\ \vdots \\ \theta_{kj}^{(l)}(t) \\ \vdots \\ \theta_{kK_l}^{(l)}(t) \end{bmatrix}$$

where l is from 1 to L-1.

• Net vector $z^{(l)}(t)$ represents all net values on the l^{th} layer at learning step t.

$$z^{(l)}(t) = \begin{bmatrix} z_1^{(l)}(t) \\ z_2^{(l)}(t) \\ \vdots \\ z_k^{(l)}(t) \\ \vdots \\ z_{K_l}^{(l)}(t) \end{bmatrix}_{K_l\times 1} = \theta^{(l-1)}(t)a^{(l-1)}(t)$$

where l is from 2 to L.

$$\begin{aligned} z_k^{(l)}(t) &= \theta_k^{(l-1)\top}(t)a^{(l-1)}(t) \\ &= \theta_{k0}^{(l-1)}(t)a_0^{(l-1)}(t) + \theta_{k1}^{(l-1)}(t)a_1^{(l-1)}(t) + \cdots + \theta_{kK_{l-1}}^{(l-1)}(t)a_{K_{l-1}}^{(l-1)}(t) \\ &= \sum_{j=0}^{K_{l-1}} \theta_{kj}^{(l-1)}(t)a_j^{(l-1)}(t) \in (-\infty, \infty) \end{aligned}$$

• The actual output vector $a^{(l)}(t)$ represents all actual output values on the l^{th}

layer at learning step t.

$$a^{(l)}(t) = \begin{bmatrix} a_1^{(l)}(t) \\ a_2^{(l)}(t) \\ \vdots \\ a_k^{(l)}(t) \\ \vdots \\ a_{K_{(l)}}^{(l)}(t) \end{bmatrix}_{K_{(l)} \times 1} = f(z^{(l)}(t)) \text{ where } f(z_k^{(l)}(t)) = \frac{1}{1+e^{\left(z_k^{(l)}(t)\right)}}$$

where l is from 2 to L.

$$a_k^{(l)}(t) = f(z_k^{(l)}(t)) \in (0,1)$$

• The selected actual output node is denoted $son(t)$ and refers to the actual output node which has the maximum actual output value among all actual output nodes on the last L^{th} layer.

$$son(t) = max_k\{a_k^{(L)}\}$$

• The reward value is denoted $reward(t)$ and it is given from the Observer. Note that $reward(t)$ with the value of 0 refers to punishment while $reward(t)$ with the value of 1 means literally reward.

$$reward(t) \in \{0, 1\}$$

• The quasi-target output vector $R(t)$ is generated based on reward value $r(t)$ as follows.

$$R(t) = \begin{bmatrix} R_1(t) \\ R_2(t) \\ \vdots \\ R_k(t) \\ \vdots \\ R_{K_L}(t) \end{bmatrix}_{K_L \times 1}$$

$$\begin{aligned} R_k(t) &\leftarrow 1 \quad \text{if } k = son(t) \text{ and } r(t) = 1 \\ R_k(t) &\leftarrow 0 \quad \text{if } k \neq son(t) \text{ and } r(t) = 1 \\ R_k(t) &\leftarrow 0 \quad \text{if } k = son(t) \text{ and } r(t) = 0 \\ R_k(t) &\leftarrow 1 \quad \text{if } k \neq son(t) \text{ and } r(t) = 0 \end{aligned}$$

• Learning input $LRBX(t)$ and learning output $LRBY(t)$ are generated as follows based on context and reward value.

$$\begin{aligned} LRBX &= context(t)^\top \quad \in \mathbb{R}^{1 \times (n+1)} \\ LRBY &= R(t)^\top \quad \in \mathbb{R}^{1 \times K_L} \end{aligned}$$

Note that context vector $context(t)$ and quasi-target output vector $R(t)$ are column vectors whereas $LRBX(t)$ and $LRBY(t)$ are row vectors in CALM-rLRB. This is because CALM-LRB-CORE shown in Algorithm 11 is based on a generalized matrixwise batch learning algorithm as shown in Algorithm 6. By having this matrixwise batch mode, CALM-LRB-CORE can take multiple input

and quasi-target output vectors in the form of a matrix in the other algorithms: CALM-eLRB, CALM-epLRB, and CALM-nepLRB. In CALM-rLRB, however, the size of $LRBX(t)$ is $1 \times (n+1)$ and the size of $LRBY(t)$ is $1 \times K_L$, which is one row vector each.

Algorithm 11 CALM-LRB-CORE Pseudocode

Given T, ITR, n, L, K_1, $\cdots$, K_L, η, λ, ϵ, γ, $LRBX(t)$, $LRBY(t)$, $\theta(t)$
$a^{(1)} = LRBX(t)$, $y = LRBY(t)$, $\theta(1) = \theta(t)$, $m = 1$
for $itr = 1$ **to** ITR **do**
  (1) FORWARD PROPAGATION
  for $l = 2$ **to** L **do**
    $z^{(l)} \leftarrow a^{(l-1)}(\theta^{(l-1)})^\top(itr)$
    $a^{(l)} \leftarrow f(z^{(l)})$
    Add $a_0^{(l)} \leftarrow +1$ for bias at each layer.
  end for
  (2) BACKPROPAGATION: ERROR UPDATE
  Remove $a_0^{(L)}$
  $\delta^{(L)} \leftarrow a^{(L)} - y$
  $\Delta\theta^{(L-1)}(itr) \leftarrow (\delta^{(L)})^\top a^{(L-1)}$
  for $l = L-1$ **to** 2 **do**
    $\delta^{(l)}(itr) \leftarrow \delta^{(l+1)}(itr)\theta^{(l)}(itr)$
    Remove $\delta_0^{(l)}(itr)$
    $\delta^{(l)} \leftarrow \delta^{(l)} .* f'(z^{(l)})$
    $\Delta\theta^{(l-1)}(itr) \leftarrow (\delta^{(l)})^\top a^{(l-1)}$
  end for
  $Cost(itr) \leftarrow \sum_{k=1}^{K_L} \left(-y_k ln(a_k^{(L)}) - (1-y_k) ln(1-a_k^{(L)})\right)$
  (3) WEIGHT UPDATE
  for $l = 1$ **to** $L-1$ **do**
    $\theta^{(l)}(itr+1) \leftarrow \theta^{(l)}(itr) - \eta\left(\frac{1}{m}\Delta\theta^{(l)}(itr) + \frac{\lambda}{m}\bar{\theta}^{(l)}(itr)\right)$
  end for
  $Reg(itr) \leftarrow \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{k=1}^{K_{l+1}} \sum_{j=1}^{K_l} \left(\theta_{kj}^{(l)}(itr)\right)^2$
  $J(itr) \leftarrow Cost(itr) + Reg(itr)$
end for
Return $\theta(ITR)$

In CALM-LRB-CORE, it takes learning input, learning output, and neural weights from CALM-rLRB-MAIN function and computes weigh updates through

the given number of iterations. At each iteration, the optimization process is roughly divided into three phases: (1) forward propagation, (2) backward propagation (backpropagation) by doing error updates, and (3) optimization by doing weight update and calculating cost values with regularization value Reg. After repeating this optimization process through the given number of iterations, ITR, it returns the updated weights after the last iteration to the main function.

Note that CALM-LRB-CORE is based on a generalized, logistic regression, matrixwise, batch learning algorithm the detailed computing processes of which are fully explained and mathematically demonstrated in Section 3.2; therefore the following paragraphs focus on how CALM-rLRB data gets through the matrixwise optimization process in each iteration rather than focusing on describing detailed computing process. Also, the notation (itr) is omitted in following mathematical steps for simplicity.

• Learning input $LRBX(t)$ and learning output $LRBY(t)$ are set as input matrix $a^{(1)}$ and quasi-target output matrix y, respectively. Then net matrix $z^{(l)}$ and actual output matrix $a^{(l)}$ at each layer are computed as follows, which is called forward propagation.

$$
\begin{aligned}
a^{(1)} &= LRBX(t); \\
y &= LRBY(t); \\
z^{(l)} &= a^{(l-1)}(\theta^{(l-1)})^{\top} \text{ where } l \text{ is from 2 to L.} \\
a^{(l)} &= f(z^{(l)}) \text{ where } l \text{ is from 2 to L.} \\
m &= 1;\ m \text{ is always 1 in CALM-rLRB}
\end{aligned}
$$

Note that, in CALM-rLRB, m is always 1 since it is does not use saved experiences, therefore $a^{(1)}$ and y are row vectors as described previously.

• The cost function $J(\theta)$ is calculated based on the value of m as follows. $a^{(1)}$ and y are row vectors thus only one summation is needed for cost value after doing elementwise multiplication of each element of $a^{(1)}$ and y.

$$J(\theta) = \frac{1}{m}\sum_{k=1}^{K_L}\left(-y_k ln(a_k^{(L)}) - (1-y_k)ln(1-a_k^{(L)})\right) + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{k=1}^{K_{l+1}}\sum_{j=1}^{K_l}\left(\theta_{kj}^{(l)}\right)^2 \tag{5.1}$$

• The error update is computed as follows, which is fully derived in Section 3.2.

$$l = L \qquad \delta^{(L)} = a^{(L)} - y \tag{5.2}$$

$$l = L-1 \; to \; 2 \qquad \delta^{(l)} = \delta^{(l+1)}\theta^{(l)} .* f'(z^{(l)}) \tag{5.3}$$

• The weight update - matrixwise is computed as follows, which is fully derived in Section 3.2.

$$l = 1 \; to \; L-1 \qquad \theta^{(l)}(t+1) = \theta^{(l)}(t) - \eta\left(\delta^{(l+1)\top}a^{(l)} + \lambda\bar{\theta}^{(l)}\right) \tag{5.4}$$

5.2.4 The Role of the Cost Function in CALM-rLRB

Similar to the role of the cost function in generalized, arbitrary-depth, ANN with logistic regression as shown in Section 3.2, $J(\theta)$ in CALM-LRB-CORE, is to provide a way of evaluating the learning effects based on the size of the error between the actual output at the last layer, $a^{(L)}$, and the quasi-target output,

$y = LRBY(t)$ which is generated by CALM-rLRB. In here, it is important to point out that $LRBY(t)$ is set by the quasi-target output $R(t)$ and $R(t)$ is based on the reward value $r(t)$; thus we can see the behavior of the cost function is based on the value of $LRBY_k(t)$ which is same as $R_k(t)$ in CALM-rLRB. Therefore, Equation (5.1) can be expanded as follows.

$$
\begin{aligned}
J(\theta) =& \frac{1}{m}\sum_{k=1}^{K_L}\Big(-y_k ln(a_k^{(L)}) - (1-y_k)ln(1-a_k^{(L)})\Big) \\
&+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{k=1}^{K_{l+1}}\sum_{j=1}^{K_l}\left(\theta_{kj}^{(l)}\right)^2 \\
=& \frac{1}{m}\sum_{k=1}^{K_L}\Big(-LRBY_k(t) ln(a_k^{(L)}) - (1-LRBY_k(t))ln(1-a_k^{(L)})\Big) \\
&+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{k=1}^{K_{l+1}}\sum_{j=1}^{K_l}\left(\theta_{kj}^{(l)}\right)^2 \\
=& \frac{1}{m}\sum_{k=1}^{K_L}(Cost_k) + Reg \qquad (5.5) \\
&\text{where } \begin{cases} Cost_k = -R_k(t) ln(a_k^{(L)}) - (1-R_k(t))ln(1-a_k^{(L)}) \\ Reg = \frac{\lambda}{2m}\sum\limits_{l=1}^{L-1}\sum\limits_{k=1}^{K_{l+1}}\sum\limits_{j=1}^{K_l}(\theta_{kj}^{(l)})^2 \end{cases}
\end{aligned}
$$

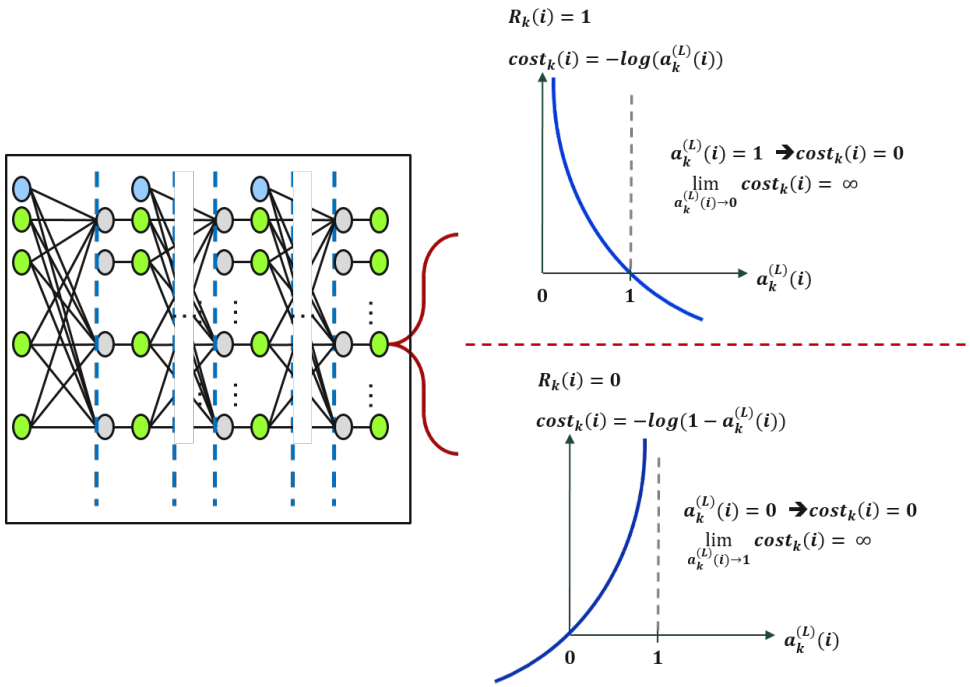


Figure 5.3: CALM-rLRB Role of the Cost Function

Figure 5.3 helps us to visually understand how the cost function evaluates its learning effects depending on its generated quasi-target output. First of all, when $LRBY_k(t)$ is 1, $Cost_k$ decreases as $a_k^{(L)}$ approaches 1; otherwise, $Cost_k$ increases since the corresponding error value increases as $a_k^{(L)}$ approaches 0. This process is described as follows.

If $LRBY_k(t) = 1$

$Cost_k = -ln(a_k^{(L)})$

$LRBY_k(t) = 1,\ a_k^{(L)} = 1 \implies \delta_k^{(L)} = 0 \implies Cost_k = 0$

$LRBY_k(t) = 1,\ a_k^{(L)} < 1 \implies \delta_k^{(L)} < 0 \implies Cost_k \uparrow$

On the other hand, when $LRBY_k(t)$ is 0, $Cost_k$ decreases as $a_k^{(L)}$ approaches 0; otherwise, $Cost_k$ increases since the corresponding error value increases as

$a_k^{(L)}$ approaches 1. This process is described as follows.

$$
\begin{aligned}
&\text{If } LRBY_k(t) = 0 \\
&Cost_k = -ln(1 - a_k^{(L)}) \\
&LRBY_k(t) = 0,\ a_k^{(L)} = 0 \implies \delta_k^{(L)} = 0 \implies Cost_k = 0 \\
&LRBY_k(t) = 0,\ a_k^{(L)} > 0 \implies \delta_k^{(L)} > 0 \implies Cost_k \uparrow
\end{aligned}
$$

In conclusion, the cost function gives smaller values if the ANN has smaller error and larger values if the ANN has larger errors. Therefore, each $Cost_k(i)$ corresponding to each quasi-target output $LRBY_k(t)$ tells us whether each actual output matches with the quasi-target output or not based on reward value.

5.2.5 The Role of Weight Update Rule in CALM-rLRB

Similar to the role of weight update in a generalized, arbitrary-depth, ANN with logistic regression as shown in Section 3.2, the weight update rule adjusts each the weight vector on each layer $\theta_k^{(l)}$ either closer to or further away from its corresponding input vector $a_k^{(l)}$. Therefore, each weight vector $\theta_k^{(l)}$ can be considered as a reward-based classifier corresponding to each actual output on the last layer $a_k^{(l+1)}$ based on $LRBY_k(t)$, which eventually decides which kind of contexts it should move closer to or move away from for each weight vector $\theta_k^{(1)}$ in the first layer. In order to look into the role of weight update in depth, we will simplify the Equation (5.4) by assuming the learning parameters as follows,

which can be divided as three possible learning cases.

$$\begin{aligned}&\text{Assume: } m = 1, \lambda = 0, \text{ and } \eta = 1\\&\text{Then: } \forall_k,\ \theta_k^{(l)}(t+1) \leftarrow \theta_k^{(l)}(t) - \delta_k^{(l+1)} a^{(l)\top}\end{aligned}$$

First, in the case of $\delta_k^{(l+1)} = 0$, which means $a_k^{(l+1)}$ is same as R_k and thus there will be no changes in updating θ_k as follows.

$$\delta_k^{(l+1)} = 0;\ (a_k^{(l+1)} = R_k) \implies \theta_k^{(l)}(t+1) = \theta_k^{(l)}(t)$$

In the cases of $\delta_k^{(l+1)} > 0$ or $\delta_k^{(l+1)} < 0$, the weight vector $\theta_k^{(l)}$ will be adjusted by moving itself closer to $a^{(l)}$ or farther away as follows.

$$\begin{cases}\delta_k^{(l+1)} > 0;\ (a_k^{(l+1)} > R_k)\\ \implies \theta_k^{(l)}(t+1) = \theta_k(t) - \delta_k a^{(l)}\\ \implies \theta_k^{(l)}(t+1) \text{ will be farther away from } a^{(l)}\\ \implies \theta_k^{(l)}(t+1)^\top a^{(l)} = z_k \text{ will be decreased}\\ \implies a_k^{(l+1)} = f(z_k) \text{ will be also decreased}\\ \implies \delta_k^{(l+1)} \text{ will be also decreased}\\ \implies Cost_k \text{ will be also decreased}\end{cases}$$

$$
\begin{cases}
\delta_k^{(l+1)} < 0;\ (a_k^{(l+1)} < R_k) \\
\implies \theta_k^{(l)}(t+1) = \theta_k(t) + \delta_k a^{(l)} \\
\implies \theta_k^{(l)}(t+1) \text{ will be closer to } a^{(l)} \\
\implies \theta_k^{(l)}(t+1)^\top a^{(l)} = z_k \text{ will be increased} \\
\implies a_k^{(l+1)} = f(z_k) \text{ will be also increased} \\
\implies \delta_k^{(l+1)} \text{ will be decreased} \\
\implies Cost_k \text{ will be also decreased}
\end{cases}
$$

5.3 CALM-eLRB

In this section, we will see how CALM-eLRB performs experience-based logistic regression neural optimization. CALM-eLRB is based on CALM-rLRB and it learns by utilizing both reward and experiences. An *experience* refers to synthetic information consisting of: (1) a context $context(t)$, (2) the corresponding selected output node $son(t)$, and (3) the reward value $r(t)$ received as a consequence of making that selection in that context. The detailed algorithm and learning process are described in their own subsections.

5.3.1 CALM-eLRB Features

The basic idea of CALM-eLRB is to utilize past positive experiences in its optimization process as well as utilizing reward information. Recall that CALM-rLRB uses $context(t)$ to set learning input $LRBX(t)$ and generates quasi-target output $R(t)$ based on reward value $r(t)$ and uses that to set learning

output $LRBY(t)$ for optimization. Building on this, CALM-eLRB saves its current experience into the Experience-based Knowledge Base (EKB) if the experience is positive. Then, during learning, not only $context(t)$ and $R(t)$ but also the saved experiences are used to generate $LRBX(t)$ and $LRBY(t)$. The concept is that even though the agent should learn based on the feedback it is receiving in its current situation, it shouldn't forget what worked for it in the past.

It is notable that there are two implications on using the phrase "past positive empirical experiences": (1) an experience is saved in the EKB only when it gets a reward value of 1 and thus (2) the number of experiences used in the optimization process can be different during each learning step, which supports incremental learning. We will first look through the overall process of CALM-eLRB and see how to save and exploit the past positive experiences for logistic regression optimization.

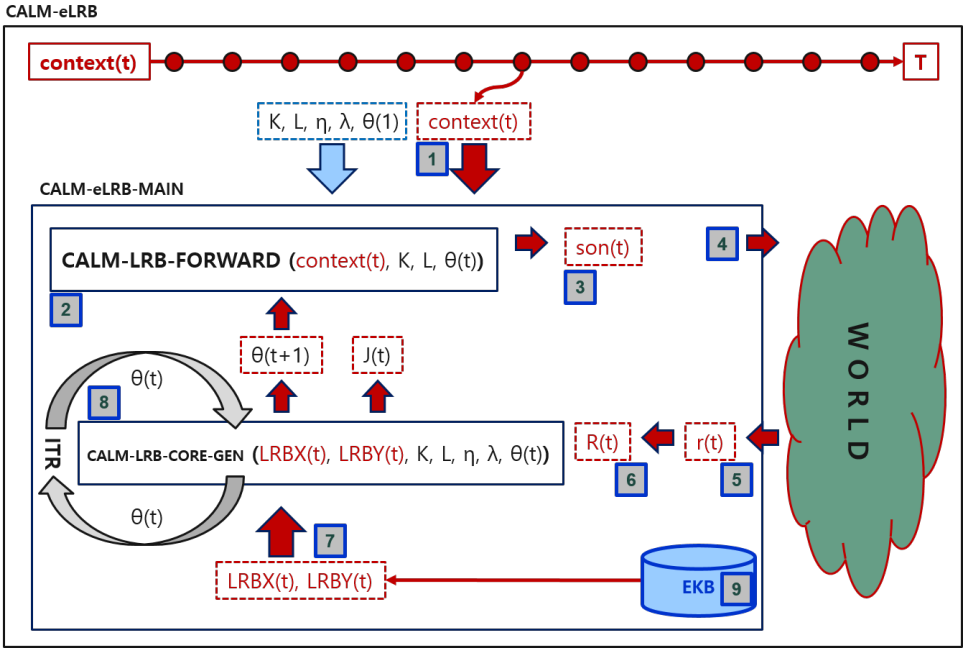


Figure 5.4: CALM-eLRB Algorithm Diagram

Figure 5.4 helps us to visually understand overall steps of CALM-eLRB; the difference from CALM-rLRB is that there is an EKB which saves rewarding experiences. Based on the CALM-rLRB learning process, CALM-eLRB has one more step since it exploits past positive experiences. The steps are as follows. (1) Take contextual input $context(t)$, (2) do forward propagation, (3) select the output node that has the maximum output value; this is denoted as the selected output node $son(t)$, (4) perform the behavioral task corresponding to the selected output node, (5) get the reward value $r(t)$ from the environment, (6) generate quasi-target output $R(t)$ based on reward value $r(t)$, (7) set learning input $LRBX(t)$ and learning output $LRBY(t)$ for the logistic regression optimization based on the contextual input $context(t)$, quasi-target output $R(t)$, and the saved experiences in the EKB, (8) optimize its current neural weights by calling the function CALM-LRB-CORE-GEN, which performs logistic regression optimization with gradient descent weight update by taking $LRBX(t)$, $LRBY(t)$, and current neural weights $\theta(t)$ as inputs, and (9) save the current experience if its reward value is positive. More specifically, in this last step, the EKB Manager takes the necessary values to generate synthetic information ($context(t)$, $son(t)$, $r(t)$) as per the definition of experience. Figure 5.4 only shows how the CALM-eLRB learning process works thus the other CALM components are omitted; however, the EKB is shown to indicate that CALM-eLRB utilizes not only the current context and reward but also saved experiences from the EKB.

Note that, compared to CALM-rLRB, only Step 7 and Step 9 are different whereas the other steps are the same.

Step	$context(t)$	$son(t)$	$r(t)$	$R(t)$	$LRBX(t)$	$LRBY(t)$	$sw(t)$
t	$[1,1,1]$	3	1	$[0,0,1]$	$[1,1,1]$	$[0,0,1]$	θ_3
$t+1$	$[1,2,2]$	3	0	$[1,1,0]$	$\begin{bmatrix}1 & 1 & 1\\ 1 & 2 & 2\end{bmatrix}$	$\begin{bmatrix}0 & 0 & 1\\ 1 & 1 & 0\end{bmatrix}$	θ_3
$t+2$	$[1,3,3]$	2	1	$[0,1,0]$	$\begin{bmatrix}1 & 1 & 1\\ 1 & 3 & 3\end{bmatrix}$	$\begin{bmatrix}0 & 0 & 1\\ 0 & 1 & 0\end{bmatrix}$	θ_2
$t+3$	$[1,4,4]$	2	1	$[0,1,0]$	$\begin{bmatrix}1 & 1 & 1\\ 1 & 3 & 3\\ 1 & 4 & 4\end{bmatrix}$	$\begin{bmatrix}0 & 0 & 1\\ 0 & 1 & 0\\ 0 & 1 & 0\end{bmatrix}$	θ_2

Table 5.3: CALM-eLRB Learning Example

For better understanding of Step 7 and Step 9, we will use the same learning used in the explanation of CALM-rLRB. Table 5.3 shows how CALM-eLRB differently generates $LRBX(t)$ and $LRBY(t)$ based on both context/reward and saved experiences.

Besides the learning example, this section introduces a new type of table named *EKB Status Table* which represents the status of the EKB during learning. An EKB status table shows how experiences are saved and accumulated over learning steps and thus each row of the table represents an experience which contains $context(t)$, $son(t)$, $r(t)$. However, note that we know $r(t)$ is always 1 in the EKB for CALM-eLRB since it only stores past rewarding experiences. Therefore, in the EKB status tables, $R(t)$ is represented instead of $r(t)$ for better understanding of the learning process. Table 5.4 and Table 5.5 show the status of the EKB at learning step t; in particular, Table 5.4 shows the EKB status before saving the current experience and Table 5.5 shows the EKB status after saving the experience at learning step t.

$past\ context(i)$	$past\ son(i)$	$past\ R(i)$
$\emptyset$	$\emptyset$	$\emptyset$

Table 5.4: CALM-eLRB EKB Status at Learning Step t Before Saving Current Experience

$past\ context(i)$	$past\ son(i)$	$past\ R(i)$
$EKB_X(i) \quad = context(t)$	3	$[0, 0, 1]$

Table 5.5: CALM-eLRB EKB Status at Learning Step t After Saving Current Experience

We will see the process of CALM-eLRB staring from the first row of Table 5.3. At learning step t, the given $context(t)$ is $[1, 1, 1]$ and the selected output nodes is 3; also, it is assumed that the selected output 3 acquires positive feedback thus $r(t)$ is set to 1. Based on this, the quasi-target output $R(t)$ corresponding to the $context(t)$ is generated as $[0, 0, 1]$ as explained in CALM-rLRB. After generating the quasi-target output, it sets learning input $LRBX(t)$ and learning output $LRBY(t)$ for the optimization based on $context(t)$, $R(t)$, and past positive experiences. However, in the learning step t, there is no past saved experiences as seen in the Table 5.4; so $LRBX(t)$ is set to be the current context $[1, 1, 1]$ and $LRBY(t)$ is assigned from the quasi-target output $R(t)$, which is same as in CALM-rLRB. After optimization, CALM-eLRB saves the current experience into the EKB since it received a positive reward; consequently, the EKB status is changed from Table 5.4 to Table 5.5.

Note that the indicator of each experience in the EKB is denoted i and the number of experiences in the EKB is denoted m; thus i can be from 0 up to m since it can happen that the EKB has no experiences. This implies that the total number of saved experiences (m) are less than or equal to the total number of

learning steps (T) because the EKB only stores successful experiences. Similarly, i is always less than or equal to the value of t at learning step t since it takes context input iteratively.

past context(i)	*past son*(i)	*past* $R(i)$
$EKB_X(i) \quad = context(t)$	3	$[0, 0, 1]$

Table 5.6: CALM-eLRB EKB Status at Learning Step t+1 Before Saving Current Experience

past context(i)	*past son*(i)	*past* $R(i)$
$EKB_X(i) \quad = context(t)$	3	$[0, 0, 1]$

Table 5.7: CALM-eLRB EKB Status at Learning Step t+1 After Saving Current Experience

In learning step $t + 1$ in Table 5.3, the given $context(t + 1)$ is $[1, 2, 2]$ with the selected output node as 3 again, but it is assumed that it gets a reward of 0 and thus the quasi-target output $R(t)$ corresponding to the $context(t + 1)$ is set to $[1, 1, 0]$. In this case, the goal of CALM-eLRB is to make the neural network able to learn that the selected output node 3 is not good for the current context $[1, 2, 2]$, but good for the past context $[1, 1, 1]$. In order to achieve this goal, CALM-eLRB applies both current experience and the past experience from the EKB in setting $LRBX(t)$ and $LRBY(t)$. If we look at Table 5.6, there is one saved past positive experience. In generating $LRBX(t + 1)$, CALM-eLRB places all past contexts first and then adds the current context last; similarly, it takes all past quasi-target output and the current quasi-target output in generating $LRBY(t + 1)$. The generated $LRBX(t + 1)$ and $LRBY(t + 1)$, are shown in Table 5.3 when the learning step is $t + 1$. By having this learning input and output in the optimization process, CALM-eLRB learns that when it

encounters the context $[1, 1, 1]$ the appropriate corresponding output is 3; but for the context $[1, 2, 2]$ the desired output should be 2 or 1, definitely not 3. After finishing the optimization process, it saves the current experience if it received a positive reward; however for the current context $context(t + 1)$, it did not acquire the positive reward so the EKB before and after saving the current experience at learning step $t+1$ are exactly same as shown in Figure 5.6 and Figure 5.7.

$past\ context(i)$	$past\ son(i)$	$past\ R(i)$
$EKB_X(i) \quad = context(t)$	3	$[0, 0, 1]$

Table 5.8: CALM-eLRB EKB Status at Learning Step t+2 Before Saving Current Experience

$past\ context(i)$	$past\ son(i)$	$past\ R(i)$
$EKB_X(i) \quad = context(t)$	3	$[0, 0, 1]$
$EKB_X(i + 1) = context(t + 2)$	2	$[0, 1, 0]$

Table 5.9: CALM-eLRB EKB Status at Learning Step t+2 After Saving Current Experience

In learning step $t+2$ in Table 5.3, the given current context is $[1, 3, 3]$ and the selected output node is 2 with a positive reward. In this case, the quasi-target output $R(t)$ is set to $[0, 1, 0]$. CALM-eLRB looks up the EKB as shown Table 5.8 and checks if there are saved past experiences. A in learning step $t + 1$, there is one saved experience so it brings it together with the current information to generate $LRBX(t + 2)$ and $LRBY(t + 2)$. The generated $LRBX(t + 2)$ and $LRBY(t + 2)$ are as shown in Table 5.3 at learning step $t + 2$. With this, CALM-eLRB optimizes its neural weights based on two experiences, one previous and one current: (1) context $[1, 1, 1]$ with desired output node 3 and (2) context $[1, 3, 3]$ with desired output 2. After this, it saves the current experience into

EKB since it received a positive reward at learning step $t+2$ so the EKB status is changed from Table 5.8 to Table 5.9.

past context(i)	*past son*(i)	*past* $R(i)$
$EKB_X(i) \quad = context(t)$	3	$[0,0,1]$
$EKB_X(i+1) = context(t+2)$	2	$[0,1,0]$

Table 5.10: CALM-eLRB EKB Status at Learning Step t+3 Before Saving Current Experience

past context(i)	*past son*(i)	*past* $R(i)$
$EKB_X(i) \quad = context(t)$	3	$[0,0,1]$
$EKB_X(i+1) = context(t+2)$	2	$[0,1,0]$
$EKB_X(i+2) = context(t+3)$	2	$[0,1,0]$

Table 5.11: CALM-eLRB EKB Status at Learning Step t+3 After Saving Current Experience

Finally, in learning step $t+3$ in Table 5.3, the given context is $[1,4,4]$ and the selected output node is 2 with positive reward. In this case, the EKB has two saved experiences as shown Table 5.10, thus $LRBX(t+3)$ and $LRBY(t+3)$ are set as shown in Table 5.3. Similarly, in the optimization process, having these inputs and outputs implies that: (1) when it encounters a context such as $[1,1,1]$ the desired output is 3, (2) when it encounters a context such as $[1,3,3]$ or $[1,4,4]$ the desired output node is 2. In other words, we can expect that context $[1,1,1]$ is classified to output node 3 and the others are classified to 2 based on the generated quasi-target output. This is the principle of CALM-eLRB which performs experience-based classification. After the optimization process, the current experience is saved as shown Table 5.11 which will be used in next learning step.

In summary, CALM-eLRB is based on CALM-rLRB, which exploits not only current reward information but also past positive experiences in generating learning input and learning output for the optimization. Therefore, it is expected that the neural network will adapt to its environment faster and better using CALM-eLRB with its EKB than using CALM-rLRB which only looks at its current experience; this is the reason why CALM-eLRB is considered a more advanced algorithm than CALM-rLRB. The performances of each are evaluated and discussed in Chapter 6.

5.3.2 CALM-eLRB-ANN

Like that CALM-ANN in CALM-rLRB, the neural network for CALM-eLRB is the same as the generalized, arbitrary-depth, ANN shown in Figure 3.17 in Section 3.2.

5.3.3 CALM-eLRB Learning

In this section, we will see how CALM-eLRB performs experience-based learning iteratively. There are two primary functions named: (1) CALM-eLRB-MAIN and (2) CALM-LRB-CORE-GEN where CALM-LRB-CORE-GEN is similar to CALM-LRB-CORE in CALM-rLRB; CALM-LRB-CORE-GEN is a generalized version of CALM-LRB-CORE since the number rows of $LRBX(t)$ is no longer always 1 in CALM-eLRB; the detail difference is described in the following paragraphs. Algorithm 12 shows the main flow of CALM-eLRB and Algorithm 13 shows the generalized version of CALM-LRB-CORE.

Algorithm 12 CALM-eLRB-MAIN Pseudocode

Given $T, ITR, n, L, K_1, \cdots, K_l, \cdots, K_L, \eta, \lambda, \epsilon, \gamma$
for $l = 1$ **to** $L - 1$ **do**
  Init $\theta^{(l)} \in R^{K_{l+1} \times (K_l+1)}$, $\theta^{(l)}(1) \leftarrow \theta^{(l)}$
end for
for $t = 1$ **to** T **do**
  (1) CONTEXT ACQUISITION
  $context(t) \in \mathbb{R}^{n \times 1}$
  Add bias for a context, $context(t) \in \mathbb{R}^{(n+1) \times 1}$
  (2) FORWARD PROPAGATION
  $a^{(1)}(t) \leftarrow context(t)$
  for $l = 2$ **to** L **do**
    $z^{(l)}(t) \leftarrow \theta^{(l-1)}(t) a^{(l-1)}(t)$
    $a^{(l)}(t) \leftarrow f(z^{(l)}(t))$
    Add $a_0^{(l)}(t) \leftarrow +1$ for bias at each layer.
  end for
  (3) OUTPUT SELECTION
  $son(t) \leftarrow max_k\{a_k^{(L)}\}$
  $selected\_weight(t) \leftarrow \theta^{(L)}_{son(t)}(t)$
  (4) BEHAVIORAL TASK
  (5) REWARD ACQUISITION
  Get $r(t) \in \{0, 1\}$ from environment
  (6) SET UP $R(t) \in \mathbb{R}^{K \times 1}$ based on $r(t)$
  if $r(t) = 1$ **then**
    $R_k(t) \leftarrow 1$, if $k = son(t)$
    $R_k(t) \leftarrow 0$, if $k \neq son(t)$
  else if $r(t) = 0$ **then**
    $R_k(t) \leftarrow 0$, if $k = son(t)$
    $R_k(t) \leftarrow 1$, if $k \neq son(t)$
  end if
  (7) SET UP $LRBX(t)$ AND $LRBY(t)$
  $m \leftarrow sizeof(EKB, 1)$; number of rows of EKB
  $LRBX(t) \leftarrow [EKB_X;\ context(t)^\top] \in \mathbb{R}^{(m+1) \times (n+1)}$
  $LRBY(t) \leftarrow [EKB_R;\ R(t)^\top] \in \mathbb{R}^{(m+1) \times K}$
  (8) CALL CALM-LRB-CORE-GEN OPTIMIZATION
  $[J(t), \theta(t+1)] \leftarrow$ CALM-LRB-CORE-GEN ($LRBX(t)$, $LRBY(t)$, $\theta(t)$)
  (9) SAVE CURRENT EXPERIENCE INTO EKB
  if $r(t) = 1$ **then**
    $EKB \leftarrow add(context(t), son(t), r(t), selected\_weight(t))$
  end if
end for

CALM-eLRB-MAIN has one more step than CALM-rLRB-MAIN: (9) save current experience into EKB if it acquired positive reward. Note that the computations for forward propagation, output selection, reward acquisition, and generating output are the same as for CALM-rLRB; thus we will see how learning input $LRBX(t)$ and learning output $LRBY(t)$ are generated in CALM-eLRB.

• Learning input $LRBX(t)$ and learning output $LRBY(t)$ are generated as follows based on past positive experiences, as well as the current context, and reward value.

$m \leftarrow sizeof(EKB, 1)$; number of rows of EKB

$EKB_X \in \mathbb{R}^{(m)\times(n+1)}$

$EKB_R \in \mathbb{R}^{(m)\times K}$

$LRBX(t) \leftarrow [EKB_X;\ context(t)^\top] \quad \Longrightarrow \quad LRBX(t) \in \mathbb{R}^{(m+1)\times(n+1)}$

$LRBY(t) \leftarrow [EKB_R;\ R(t)^\top] \quad \Longrightarrow \quad LRBY(t) \in \mathbb{R}^{(m+1)\times K}$

Algorithm 13 CALM-LRB-CORE-GEN Pseudocode

Given T, ITR, n, L, K_1, $\cdots$, K_L, η, λ, ϵ, γ, $LRBX(t)$, $LRBY(t)$, $\theta(t)$
$a^{(1)} = LRBX(t)$, $y = LRBY(t)$, $\theta(1) = \theta(t)$, $m = sizeof(y)$
for $itr = 1$ **to** ITR **do**
  (1) FORWARD PROPAGATION
  for $l = 2$ **to** L **do**
    $z^{(l)} \leftarrow a^{(l-1)}(\theta^{(l-1)})^\top(itr)$
    $a^{(l)} \leftarrow f(z^{(l)})$
    Add $a_0^{(l)} \leftarrow +1$ for bias at each layer.
  end for
  (2) BACKPROPAGATION: ERROR UPDATE
  Remove $a_0^{(L)}$
  $\delta^{(L)} \leftarrow a^{(L)} - y$
  $\Delta\theta^{(L-1)}(itr) \leftarrow (\delta^{(L)})^\top a^{(L-1)}$
  for $l = L - 1$ **to** 2 **do**
    $\delta^{(l)} \leftarrow \delta^{(l+1)}\theta^{(l)}(itr)$
    Remove $\delta_0^{(l)}$
    $\delta^{(l)} \leftarrow \delta^{(l)} .* f'(z^{(l)})$
    $\Delta\theta^{(l-1)}(itr) \leftarrow (\delta^{(l)})^\top a^{(l-1)}$
  end for
  if $m = 1$ **then**
    $Cost(itr) \leftarrow \sum_{k=1}^{K_L} \left(-y_k ln(a_k^{(L)}) - (1 - y_k) ln(1 - a_k^{(L)})\right)$
  else
    $Cost(itr) \leftarrow \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K_L} \left(-y_k(i) ln(a_k^{(L)}(i)) - (1 - y_k(i)) ln(1 - a_k^{(L)}(i))\right)$
  end if
  (3) WEIGHT UPDATE
  for $l = 1$ **to** $L - 1$ **do**
    $\theta^{(l)}(itr + 1) \leftarrow \theta^{(l)}(itr) - \eta \left(\frac{1}{m}\Delta\theta^{(l)}(itr) + \frac{\lambda}{m}\bar{\theta}^{(l)}(itr)\right)$
  end for
  $Reg(itr) \leftarrow \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{k=1}^{K_{l+1}} \sum_{j=1}^{K_l} \left(\theta_{kj}^{(l)}(itr)\right)^2$
  $J(itr) \leftarrow Cost(itr) + Reg(itr)$
end for
Return $\theta(ITR)$

Similar to CALM-LRB-CORE, CALM-LRB-CORE-GEN also takes learning input, learning output, and neural weights from CALM-eLRB-MAIN and does weigh updates through the given number of iterations. The difference is that m is not always 1 while it was always 1 in CALM-LRB-CORE. This is because

$LRBX(t)$ and $LRBY(t)$ are generated based not only on the current context and reward but also on experiences in the EKB. These are generated in CALM-eLRB-MAIN and are passed to CALM-LRB-CORE-GEN. CALM-LRB-CORE-GEN is used by CALM-eLRB, CALM-epLRB, and CALM-nepLRB, all of which use the EKB, while CALM-LRB-CORE is used only by CALM-rLRB.

Note that CALM-LRB-CORE-GEN also takes all the learning parameters: T, ITR, n, L, K_1, $\cdots$, K_l, $\cdots$, K_L, η, λ, ϵ, γ as well as $LRBX(t)$, $LRBY(t)$, and $\theta(t)$; for simplicity, all parameters are omitted from the pseudocode of Algorithm 12.

CALM-LRB-CORE-GEN is also a generalized logistic regression matrixwise batch learning algorithm and the detailed computing processes of it are fully explained and mathematically demonstrated in Section 3.2; therefore the following paragraphs focus on how CALM-eLRB data gets through the matrixwise optimization process rather than focusing on describing the detailed computing processes.

• Learning input $LRBX(t)$ and learning output $LRBY(t)$ are set as input matrix $a^{(1)}$ and quasi-target output matrix y. Then net matrix $z^{(l)}$ and actual output matrix $a^{(l)}$ at each layer are computed as follows, which is called forward

propagation.

$$
\begin{aligned}
a^{(1)} &= LRBX(t) \in \mathbb{R}^{(m+1)\times(n+1)} \\
a^{(1)} &= \begin{bmatrix} a^{(1)}(1)^\top \longrightarrow \\ a^{(1)}(2)^\top \longrightarrow \\ \vdots \\ a^{(1)}(i)^\top \longrightarrow \\ \vdots \\ a^{(1)}(m+1)^\top \longrightarrow \end{bmatrix}_{(m+1)\times(K_1+1)} = \begin{bmatrix} LRBX(1)^\top \longrightarrow \\ LRBX(2)^\top \longrightarrow \\ \vdots \\ LRBX(i)^\top \longrightarrow \\ \vdots \\ LRBX(m+1)^\top \longrightarrow \end{bmatrix}_{(m+1)\times(K_1+1)} \\
z^{(l)} &= a^{(l-1)}(\theta^{(l-1)})^\top \text{ where } l \text{ is from 2 to L.} \\
z^{(l)} &= \begin{bmatrix} z^{(l)}(1)^\top \longrightarrow \\ z^{(l)}(2)^\top \longrightarrow \\ \vdots \\ z^{(l)}(i)^\top \longrightarrow \\ \vdots \\ z^{(l)}(m+1)^\top \longrightarrow \end{bmatrix}_{(m+1)\times K_l} = \begin{bmatrix} a^{(l-1)}(1)(\theta^{(l-1)})^\top \\ a^{(l-1)}(2)(\theta^{(l-1)})^\top \\ \vdots \\ a^{(l-1)}(i)(\theta^{(l-1)})^\top \\ \vdots \\ a^{(l-1)}(m+1)(\theta^{(l-1)})^\top \end{bmatrix}_{(m+1)\times K_l} \\
&= a^{(l-1)}(\theta^{(l-1)})^\top \\
a^{(l)} &= f(z^{(l)}) \text{ where } l \text{ is from 2 to L.} \\
a^{(l)} &= \begin{bmatrix} a^{(l)}(1)^\top \longrightarrow \\ a^{(l)}(2)^\top \longrightarrow \\ \vdots \\ a^{(l)}(i)^\top \longrightarrow \\ \vdots \\ a^{(l)}(m+1)^\top \longrightarrow \end{bmatrix}_{(m+1)\times(K_l+1)} \\
m &= size(y, 1); \text{ number of rows of } y
\end{aligned}
$$

• The cost function $J(\theta)$ is calculated as follows. Note that $LRBX$ and $LRBY$ are no longer row vectors; they represent $m+1$ learning inputs and learning outputs and thus the computation for the cost function is specified in the following form. If m is 1, the computation is the same as in Equation (5.1). On the other hand, if m is greater than 1, $a^{(1)}$ and y are matrices thus one more summation is needed in order to sum each cost value of each element in each matrix.

$$\begin{aligned} J(\theta) =& \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K_L}\Big(-y_k(i)ln(a_k^{(L)}(i)) - (1-y_k(i))ln(1-a_k^{(L)}(i))\Big) \\ &+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{k=1}^{K_{l+1}}\sum_{j=1}^{K_l}\Big(\theta_{kj}^{(l)}\Big)^2 \end{aligned} \tag{5.6}$$

• Error Update - matrixwise.

$$\begin{aligned} l = L \qquad & \delta^{(L)} = a^{(L)} - LRBY \\ l = L-1 \; to \; 2 \qquad & \delta^{(l)} = \delta^{(l+1)}\theta^{(l)} \\ & = \begin{bmatrix} \delta^{(l+1)}(1)^\top\theta^{(l)} \longrightarrow \\ \delta^{(l+1)}(2)^\top\theta^{(l)} \longrightarrow \\ \vdots \\ \delta^{(l+1)}(i)^\top\theta^{(l)} \longrightarrow \\ \vdots \\ \delta^{(l+1)}(m)^\top\theta^{(l)} \longrightarrow \end{bmatrix}_{m\times K_l} \\ & \delta^{(l)} = \sum_{i=1}^{m}\sum_{k=1}^{K_l}\Big(\delta_{ik}^{(l)} f'(z_{ik}^{(l)})\Big) \end{aligned} \tag{5.7}$$

• Weight Update - matrixwise.

$$
\begin{aligned}
& a^{(1)} = LRBX \\
& l = 1 \; to \; L-1 \qquad \theta^{(l)}(t+1) = \theta^{(l)}(t) - \eta \left(\frac{1}{m} \delta^{(l+1)\top} a^{(l)} + \frac{\lambda}{m} \bar{\theta}^{(l)} \right)
\end{aligned}
\tag{5.8}
$$

5.4 CALM-epLRB

In this section, the concept of CALM-epLRB is explained including its motivation, the detailed algorithm, and the learning effects. CALM-epLRB is based on CALM-eLRB and it learns by using rewards, extended experiences, and a new concept for a learning rule named *Selective-Power-Update (SPU)*. In naming of CALM-epLRB, "experience-powered (ep)" itself is named to indicate CALM-epLRB utilizes these extended experiences and applies the Selective-Power-Update in its learning process.

Selective-Power-Update is a newly added weight update rule which adjusts current neural connection weights using connection weights stored in the EKB as part of an extended experience. The selection of an appropriate extended experience is independent of the currently selected output node $son(t)$ and reward $r(t)$, but is dependent on a selected output node $son(i)$ in the EKB relevant to the current context $context(t)$.

Recall that CALM-rLRB only uses current context and reward information and CALM-eLRB uses both current information and experiences from the EKB. Compared to CALM-eLRB, CALM-epLRB not only uses current and past experiences but also uses extended experiences through the Selective-Power-Update

rule in order to improve the learning effects from its own experiences. CALM-epLRB has two types of weight update rule where the first is experience-based neural optimization, the same as in CALM-epLRB and the second is Selective-Power-Update. In each learning step, CALM-epLRB performs experience-based neural optimization first and then applies Selective-Power-Update. However, Selective-Power-Update can be performed only when there is a context in the EKB that is similar (or identical) to the current context. If there is no similar (or identical) context in the EKB, CALM-epLRB applies only experience-based optimization. In this way, CALM-epLRB supports experience-powered learning optimization.

5.4.1 CALM-epLRB Features

The basic idea of Selective-Power-Update is to find a rewarding neural connection related to a past context that is similar to the current context and to use the connection weights stored for it in order to update the current neural network. Selective-Power-Update is named because it updates current neural network selectively using the "power" of past rewarding weights. For example, in *context A*, if the current neural network set $son(t)$ to be *output* 1 but in the EKB the stored selected output node is *output* 2 for the stored context most similar to *context A*; then Selective-Power-Update retrieves the corresponding saved weights from the EKB and uses them to update the current neural weights. In this way, it is expected that *output* 2 will have a greater chance to be selected in the future if a similar context occurs. To briefly summarize, Selective-Power-Update (1) finds the most similar context in the EKB based on the current context, (2) retrieves the rewarding neural connection between the similar context and the corresponding selected output node in the EKB, and (3) adjusts the

current neural weight based on the retrieved neural connection.

The goal of applying Selective-Power-Update is to increase the learning speed by utilizing the past successful neural connections when the past context is similar to the current context. Therefore, the bottom line of CALM-epLRB is how to modify the current neural weight based on the past neural weight. The detailed process of Selective-Power-Update is described in Section 5.4.3.

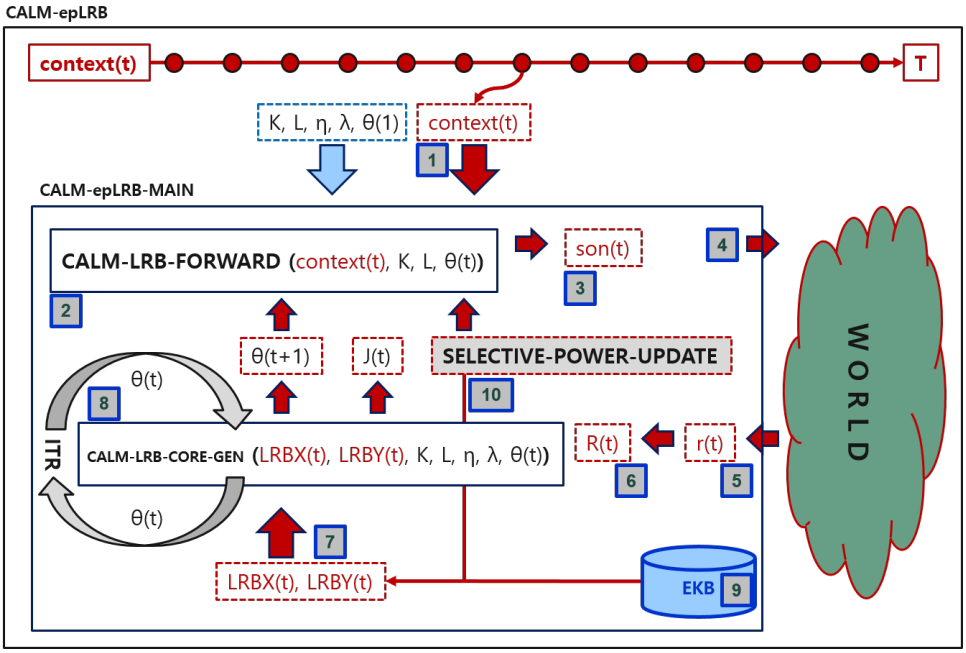


Figure 5.5: CALM-epLRB Algorithm Diagram

Figure 5.5 helps us to visually understand the overall steps of CALM-epLRB; compared to CALM-eLRB, CALM-epLRB has one more step for doing Selective-Power-Update. In addition, the experiences saved are extended experiences. The steps are as follows. (1) Take contextual input $context(t)$, (2) do forward propagation, (3) select the output node that has the maximum output value; this is denoted as the selected output node $son(t)$, (4) perform the behavioral task corresponding to the selected output node, (5) get the reward value $r(t)$

from the environment, (6) generate quasi-target output $R(t)$ based on reward value $r(t)$, (7) set learning input $LRBX(t)$ and learning output $LRBY(t)$ for the logistic regression optimization based on the contextual input $context(t)$, quasi-target output $R(t)$, and the saved experiences in the EKB, (8) optimize its current neural weights by calling the function CALM-LRB-CORE-GEN, which performs logistic regression optimization by taking $LRBX(t)$, $LRBY(t)$, and current neural weights $\theta(t)$ as inputs, (9) save the current experience if the reward value is positive, and (10) perform Selective-Power-Update.

More specifically, after finishing the experience-based optimization for given a current context, Selective-Power-Update has the following specific step. (1) Check if there is a sufficiently similar past context in the EKB compared to the current context, (2) if there is a sufficiently similar context, chose the most similar such context, (3) retrieve the past selected output node corresponding to the chosen past context, (4) retrieve the past selected weights associated with the chosen past context, and then (5) apply the past selected weights to the current neural network by adding them and the current neural weights which are related to the past selected output node.

Note that it is always possible that the past selected output node from the EKB is different from the currently selected output node. This means that, although the past context and the current context are similar, the output for each is different; in this case, Selective-Power-Update adjusts the current neural network such that it will be more likely to select the past selected output node by modifying the current neural weights to be more similar corresponding to those that selected the past selected output node. In other words, Selective-Power-Update is a vector summation of the past weight vector and the current

weight vector where both of them are related to the past selected output node.

The big difference from CALM-eLRB is that CALM-epLRB uses not only past positive experiences but also past neural connections between the past context (which is similar to current context) and the past selected output. The Selective-Power-Update is motivated from the CARB-LM [56].

It is important to mention that the phrase "the most similar" implies there might be several past contexts which are similar to the current context. In this regard, CALM-epLRB utilizes a new learning parameter, a *similarity rate* denoted as ϵ, which is used to compute similarity between two contexts; it acts as a threshold in finding similar contexts in the EKB. The metric used for measuring the similarity between two contexts is a simple *Euclidean* distance. Also, there is another learning parameter, denoted as γ, which is the *selective power rate* deciding how much to apply the past neural weight to the current neural weight when merging two weights. The equation is described in Section 5.4.3.

Step	$context(t)$	$son(t)$	$r(t)$	$R(t)$	$LRBX(t)$	$LRBY(t)$	$sw(t)$
t	$[1,1,1]$	3	1	$[0,0,1]$	$[1,1,1]$	$[0,0,1]$	θ_3
$t+1$	$[1,2,2]$	3	0	$[1,1,0]$	$\begin{bmatrix}1&1&1\\1&2&2\end{bmatrix}$	$\begin{bmatrix}0&0&1\\1&1&0\end{bmatrix}$	θ_3
$t+2$	$[1,3,3]$	2	1	$[0,1,0]$	$\begin{bmatrix}1&1&1\\1&3&3\end{bmatrix}$	$\begin{bmatrix}0&0&1\\0&1&0\end{bmatrix}$	θ_2
$t+3$	$[1,4,4]$	2	1	$[0,1,0]$	$\begin{bmatrix}1&1&1\\1&3&3\\1&4&4\end{bmatrix}$	$\begin{bmatrix}0&0&1\\0&1&0\\0&1&0\end{bmatrix}$	θ_2
$t+4$	$[1,0,1]$	2	0	$[1,0,1]$	$\begin{bmatrix}1&1&1\\1&3&3\\1&4&4\\1&0&1\end{bmatrix}$	$\begin{bmatrix}0&0&1\\0&1&0\\0&1&0\\1&0&1\end{bmatrix}$	θ_2

Table 5.12: CALM-epLRB Learning Example Before Selective-Power-Update

past context(i)	*past son*(i)	*past* $R(i)$	*past sw*(i)
$EKB_X(i) \quad = context(t)$	3	$[0,0,1]$	$EKB_\theta(i)$
$EKB_X(i+1) = context(t+2)$	2	$[0,1,0]$	$EKB_\theta(i+1)$
$EKB_X(i+2) = context(t+3)$	2	$[0,1,0]$	$EKB_\theta(i+2)$

Table 5.13: CALM-epLRB EKB Status at Learning Step t+4 Before and After Saving Current Experience

For better understanding, we will use same simple learning example in CALM-eLRB with one more example as shown on learning step $t+4$ in Table 5.12. It shows how CALM-epLRB generates its learning input and output based on the given contexts and rewards; and those are same process in CALM-eLRB. Note that Table 5.12 shows CALM-epLRB learning example before performing SPU while Table 5.14 shows CALM-epLRB learning example after performing SPU. Table 5.13 shows the EKB status when learning step is t+4 along with Table 5.12.

Step	*context*(t)	*son*(t)	$r(t)$	$R(t)$	$LRBX(t)$	$LRBY(t)$	$sw(t)$
t	$[1,1,1]$	3	1	$[0,0,1]$	$[1,1,1]$	$[0,0,1]$	θ_3
$t+1$	$[1,2,2]$	3	0	$[1,1,0]$	$\begin{bmatrix}1&1&1\\1&2&2\end{bmatrix}$	$\begin{bmatrix}0&0&1\\1&1&0\end{bmatrix}$	θ_3
$t+2$	$[1,3,3]$	2	1	$[0,1,0]$	$\begin{bmatrix}1&1&1\\1&3&3\end{bmatrix}$	$\begin{bmatrix}0&0&1\\0&1&0\end{bmatrix}$	θ_2
$t+3$	$[1,4,4]$	2	1	$[0,1,0]$	$\begin{bmatrix}1&1&1\\1&3&3\\1&4&4\end{bmatrix}$	$\begin{bmatrix}0&0&1\\0&1&0\\0&1&0\end{bmatrix}$	θ_2
$t+4$	$[1,0,1]$	3	0	$[1,0,1]$	$\begin{bmatrix}1&1&1\\1&3&3\\1&4&4\\1&0&1\end{bmatrix}$	$\begin{bmatrix}0&0&1\\0&1&0\\0&1&0\\1&0&1\end{bmatrix}$	θ_3

Table 5.14: CALM-epLRB Learning Example After Selective-Power-Update

In the learning step $t+4$ in Table 5.12, we can see the current selected output node at learning step $t+4$ is 2 and it gets zero reward; thus the experience-based optimization makes θ_2 farther away from $context(t+4)$, which is same

process as in CALM-eLRB. After the optimization, CALM-epLRB performs Selective-Power-Update by looking for past positive experiences in the EKB. In this case, it important to point out that the $context(t+4)$ is similar to $context(t)$ which is saved as $EKB_X(i)$ as shown Table 5.13. CALM-epLRB finds $EKB_X(i)$ which is similar to $context(t+4)$ and retrieves both past successfully selected output node 3 and the corresponding selected neural weight $EKB_\theta(i)$; then it modifies current neural weight vector which is corresponding to actual output 3 as follows: $\theta_3(t+5) = (1-\gamma)\theta_3(t+5) + (\gamma)EKB_\theta(i)$. Note that in the equation the learning step is $t+5$, not $t+4$. This is because θ is already updated from $\theta(t+4)$ to $\theta(t+5)$ in experience-based optimization as shown in Figure 5.5 and Selective-Power-Update is performed after that. This Selective-Power-Update expects to lead current neural network to power selecting the past successful output node 3, instead of current false selected output node 2.

More specifically, Table 5.14 shows the expected learning table after the Selective-Power-Update of CALM-epLRB based on the assumption: γ is 1. We can expect $son(t+4)$ is changed from 2 to 3 and $sw(t+4)$ is changed from θ_2 to θ_3. This can be expected by two factors: (1) θ_2 made farther away from $context(t+4)$ in experience-based optimization due to the zero reward and (2) Selective-Power-Update made θ_3 closer to $context(t+4)$ since the γ is assumed as 1. In this way, if $context(t+5)$ occurs which is similar to $context(t+4)$ at next learning step, θ_3 will be selected with positive reward based on expected learning table as shown Table 5.14. Note that Table 5.14 is expected learning example results after performing Selective-Power-Update and thus there are no changes except for $son(t+4)$ and $sw(t+4)$. This is because Selective-Power-Update does not affect the rewards, learning input, and learning output which were already generated after forward propagation. It is performed at the end of the learning process as shown in Figure 5.5 and thus it only have impact on chang-

ing selected neural connections. The detailed process is described in Section 5.4.3.

In summary, CALM-epLRB selectively gives the power to current neural weight by utilizing the past successful neural weight based on the similar context. This expect to increase the learning speed than only using the typical logistic regression optimization.

5.4.2 CALM-epLRB-ANN

Like that CALM-ANN in CALM-rLRB and CALM-eLRB, the neural network for CALM-epLRB is also same as the generalized, arbitrary-depth, ANN shown in Figure 3.17 in Section 3.2.

5.4.3 CALM-epLRB Learning

In this section, we will see how CALM-epLRB performs experience-based neural context-awareness iteratively. There are three primary functions named: (1) CALM-epLRB-MAIN, (2) CALM-LRB-CORE-GEN where CALM-LRB-CORE-GEN is same as in CALM-eLRB, and (3) CALM-SELECTIVE-POWER-LEARNING. Algorithm 14 shows the main flow of CALM-epLRB.

Algorithm 14 CALM-epLRB-MAIN Pseudocode

Given $T, ITR, n, L, K_1, \cdots, K_l, \cdots, K_L, \eta, \lambda, \epsilon, \gamma$
for $l = 1$ **to** $L - 1$ **do**
  Init $\theta^{(l)} \in R^{K_{l+1} \times (K_l+1)}$, $\theta^{(l)}(1) \leftarrow \theta^{(l)}$
end for
for $t = 1$ **to** T **do**
  (1) CONTEXT ACQUISITION
  $context(t) \in \mathbb{R}^{n \times 1}$
  Add bias for a context, $context(t) \in \mathbb{R}^{(n+1) \times 1}$
  (2) FORWARD PROPAGATION
  $a^{(1)}(t) \leftarrow context(t)$
  for $l = 2$ **to** L **do**
    $z^{(l)}(t) \leftarrow \theta^{(l-1)}(t) a^{(l-1)}(t)$
    $a^{(l)}(t) \leftarrow f(z^{(l)}(t))$
    Add $a_0^{(l)}(t) \leftarrow +1$ for bias at each layer.
  end for
  (3) OUTPUT SELECTION
  $son(t) \leftarrow max_k\{a_k^{(L)}\}$
  $selected\_weight(t) \leftarrow \theta_{son(t)}^{(L)}(t)$
  (4) BEHAVIORAL TASK
  (5) REWARD ACQUISITION
  Get $r(t) \in \{0, 1\}$ from environment
  (6) SET UP $R(t) \in \mathbb{R}^{K \times 1}$ based on $r(t)$
  if $r(t) = 1$ **then**
    $R_k(t) \leftarrow 1$, if $k = son(t)$;$R_k(t) \leftarrow 0$, if $k \neq son(t)$
  else if $r(t) = 0$ **then**
    $R_k(t) \leftarrow 0$, if $k = son(t)$;$R_k(t) \leftarrow 1$, if $k \neq son(t)$
  end if
  (7) SET UP $LRBX(t)$ AND $LRBY(t)$
  $m \leftarrow sizeof(EKB, 1)$; number of rows of EKB
  $LRBX(t) \leftarrow [EKB_X;\ context(t)^\top] \in \mathbb{R}^{(m+1) \times (n+1)}$
  $LRBY(t) \leftarrow [EKB_R;\ R(t)^\top] \in \mathbb{R}^{(m+1) \times K}$
  (8) CALL CALM-LRB-CORE-GEN OPTIMIZATION
  $[J(t), \theta(t+1)] \leftarrow$ CALM-LRB-CORE-GEN ($LRBX(t)$, $LRBY(t)$, $\theta(t)$)
  (9) SAVE CURRENT EXPERIENCE INTO EKB
  if $r(t) = 1$ **then**
    $EKB \leftarrow add(context(t), son(t), r(t), selected\_weight(t))$
  end if
  (10) SELECTIVE POWER UPDATE
  if $m > 0$ **then**
    $[\theta(t+1)] =$ CALM-SELECTIVE-POWER-LEARNING($\theta(t+1)$, EKB)
  end if
end for

In CALM-epLRB-MAIN, it has one more step based on CALM-eLRB at each learning step t: (10)it performs Selective-Power-Update rule. Note that all the computations from step (1) to (9) are same as in CALM-eLRB; thus this section only focuses on the Selective-Power-Update learning, which is shown in Algorithm 15.

Algorithm 15 CALM-SELECTIVE-POWER-LEARNING Pseudocode

$C = \{EKB_X(i) \mid i \in [1, m],\ \|context(t) - EKB_X(i)\| < \epsilon\}$
if $C \neq \emptyset$ **then**
  Find p where $\|context(t) - EKB_X(p)\| = min\{\forall_{v \in C}, \|context(t) - v\|\ \}$
  $pson = EKB_{son}(p)$
  $\theta_{pson}^{(L-1)}(t+1) = (1-\gamma)\theta_{pson}^{(L-1)}(t+1) + (\gamma)EKB_\theta(p)$
end if
return $\theta(t+1)$

• Context pool C refers to a set which has similar contexts in the EKB to the current context $context(t)$. In order to calculate the similarity, CALM-epLRB checks if the Euclidean distance between two nodes is less than the similarity rate ϵ as follows. Note that $EKB_X(i)$ is i^{th} saved context in the EKB.

$$C = \{EKB_X(i) \mid i \in [1, m],\ \|context(t) - EKB_X(i)\| < \epsilon\}$$

• $EKB_X(p)$ refers to past context which is the same or most similar vector to the current context which has smallest Euclidean distance value among all the vectors in the context pool C. Also the corresponding past selected output node is denoted as $EKB_{son}(p)$ and the past selected weight vector $EKB_\theta(p)$. Note that knowledge redundancy and knowledge inconsistency are checked by Knowledge Manager in every learning step; thus if there are exact two same contexts with different selected actual output, EKB Manager will remain most

recent context in the EKB.

Find p where $\|context(t) - EKB_X(p)\| = min\{\forall_{v \in C}, \|context(t) - v\| \}$

• Selective-Power-Update rule is defined as follows. Recall that this second type of weight update rule is proceed after the experience-based, arbitrary-depth, neural optimization; thus Selective-Power-Update learning takes the updated weights $\theta(t+1)$ from the optimization and returns the selectively updated weights with same learning step index $\theta(t+1)$, which will be used in next learning step $t+1$ in the main process. Note that *pson* refers to the past selected output node.

$$\theta_{pson}^{(L-1)}(t+1) = (1-\gamma)\theta_{pson}^{(L-1)}(t+1) + (\gamma)EKB_{\theta}(p)$$

This equation implies important feature of Selective-Power-Update: Selective-Power-Update only updates the weights in the last layer if CALM-ANN has hidden layers. This means the retrieved past selected weight in the EKB is also weight vector which is associated with the past selected output node at the last layer.

5.5 CALM-nepLRB

Finally, this section introduces the principles of a novel, bio-inspired, arbitrary-depth, neural learning model including: (1) the concept of CALM-nepLRB, (2) the CALM-nep-ANN, and (3) the CALM-nepLRB algorithm and learning process.

5.5.1 CALM-nepLRB Features

CALM-nepLRB is the key algorithm for building a novel, bio-inspired, arbitrary-depth, neural learning model. The basic idea of CALM-nepLRB is to extract some feasible features of behavioral neurobiology and utilize them to improve CALM-epLRB. CALM-nepLRB is designed based on CALM-rLRB, CALM-eLRB, CALM-epLRB, and additional neurobiological features: (1) combination-sensitive neurons, (2) recurrent inhibition, (3) appetitive learning with serotonergic neuromodulation, and (4) aversive learning with dopaminergic neuromodulation. CALM-nepLRB most advanced algorithm in CALM and it outperforms the other algorithms, as demonstrated and discussed in Chapter 6. CALM-nepLRB has flexibility in selecting a learning algorithm between those used by CALM-rLRB and CALM-epLRB depending on current learning status, which allows it to outperform the other three algorithms. Each unique bio-inspired feature of CALM-nepLRB is described in depth the following paragraphs and the learning process for CALM-nepLRB is described in Section 5.5.3. Note that CALM-nepLRB has its own novel neural network, CALM-nepLRB-ANN, which provides the features mentioned above. This neural network is described in Section 5.5.2.

Combination-Sensitive Neurons As introduced in Section 2.6, combination-sensitive neurons play important roles in triggering more complex behaviors by behaving in fundamentally different ways depending on the combination of inputs they receive. CALM-nepLRB is inspired by the combination-sensitive neurons from several sources: (1) the auditory cortex of bats, (2) the external nucleus of the inferior colliculus of owls, and (3) the torus semicircularis of electric fish. A combination-sensitive neuron in CALM-nepLRB-ANN shows its

responses only when it receives two types of input signals where one is from the output nodes in the last layer (the output layer) and the other one is from the Observer with a reward value. If the reward is positive, the combination sensitive neuron shows its response by discharging serotonin; if the reward is zero, it releases dopamine. In this way, the role of combination-sensitive neurons is to provide two different ways of reacting based on reward value; this is biologically supported by the above three animal examples.

Recurrent Inhibition CALM-nepLRB uses this recurrent inhibition to set the maximum number of times that an agent will repeat the same learning behavior which resulted in consecutive zero rewards. The maximum number may be set by the system user based on the application. The learning parameter for the maximum number is denoted as IHV meaning inhibition value. See Section 5.1 for learning parameters used in this dissertation; IHV is set to be 3, meaning the robot can not repeat the same incorrect behavior over three times.

Appetitive Learning with Serotonin vs Aversive Learning with Dopamine The CALM-nepLRB neural network performs reward-based neuromodulation based on a biological model: serotonin used in appetitive learning while dopamine is used in aversive learning. In CALM-nepLRB, a combination-sensitive neuron releases serotonin when it gets positive reward but it releases dopamine when it gets zero reward. Note that the combination-sensitive neurons only release neurotransmitters when the reward input is combined with output from the neurons in last layer. This neural design is described in Section 5.5.2.

Two Effects of Dopamine CALM-nepLRB uses dopaminergic neuromodulation which was introduced in Section 2.6. Recall that Dopamine shows its effects on both appetitive learning and aversive learning depending on two types

of dopaminergic receptor. The details of applying this biological feature in CALM-nepLRB is described in Section 5.5.2.

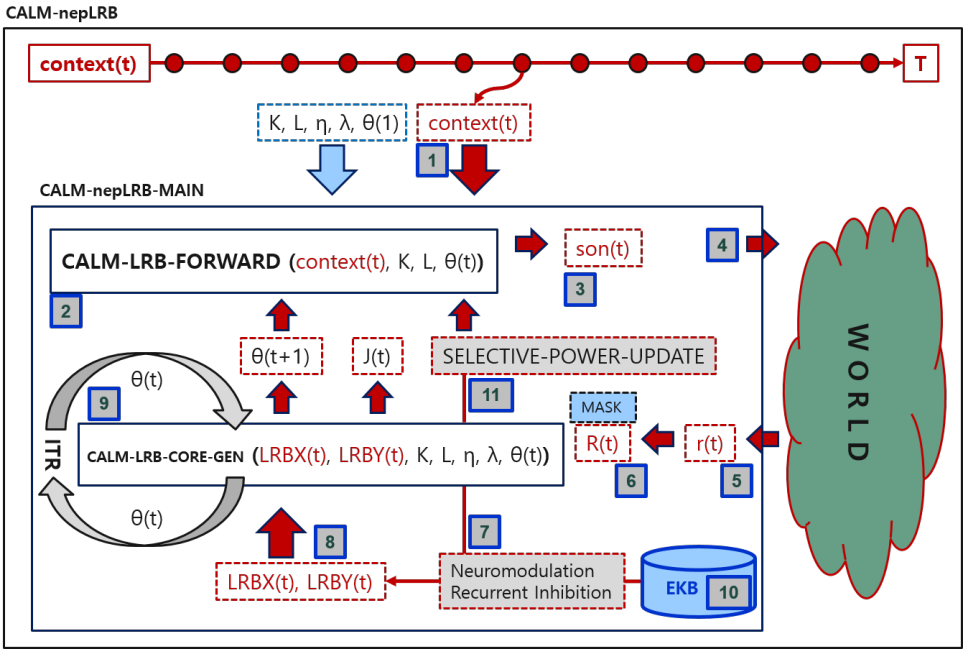


Figure 5.6: CALM-nepLRB Algorithm Diagram

Figure 5.6 helps us to visually understand the overall steps of CALM-nepLRB; compared to CALM-rLRB, CALM-eLRB, and CALM-epLRB, it is more advanced and sophisticated algorithm by having additional biological step, as follows. (1) Take context $context(t)$, (2) do forward propagation, (3) select maximum output node $son(t)$ at the last layer, (4) perform the behavioral task corresponding to the selected output node, (5) get reward $r(t)$, (6) generate the quasi-target output $R(t)$ and $MASK$ based on the reward, (7) based on $r(t)$, do neuromodulation and check recurrent inhibition for each output node, (8) set the learning input $LRBX(t)$ and output $LRBY(t)$ for the logistic regression optimization depending on the recurrent inhibition status, (9) optimize by calling the function CALM-LRB-CORE-GEN, which performs logistic regression

optimization by taking $LRBX(t)$, $LRBY(t)$, and current neural weights $\theta(t)$ as inputs, (10) save the current experience if the reward value is positive, and (11) do Selective-Power-Update if there is no recurrent inhibition.

The algorithm steps from (1) to (5) are the same as in CALM-epLRB. In CALM-nepLRB, quasi-target output $R(t)$ is generated in a more advanced way than in the other algorithms by applying another array, called $MASK$, which is same size as $R(t)$. $MASK$ keeps track of which selected output nodes have received rewards with a value of zero since the agent started to get zero rewards for its actions. The values from $MASK$ are then directly assigned to $R(t)$ so that $R(t)$ can have multiple zero elements when $r(t)$ is zero. This mechanism helps the agent to choose other behaviors that have not received zero reward for the current context. On the other hand, if the agent gets a positive reward, all the elements of $MASK$ are set to be 1 so that it can refresh previously checked zero rewards. Note that if a reward is zero, all elements of $R(t)$ except for the currently selected output are set to zero in CALM-rLRB, CALM-eLRB, and CALM-epLRB; thus the three algorithms are not able to track which output nodes have already been tried, and which have not, for this context. In this way, CALM-nepLRB is expected to find an appropriate behavior faster than the others.

After generating quasi-target output $R(t)$ with $MASK$ based on reward value, CALM-nepLRB performs neuromodulation and checks recurrent inhibition, also based on reward value, which is possible since CALM-nepLRB has a novel neural network, CALM-nepLRB-ANN, introduced in Section 5.5.2. To briefly summarize note that each output node is connected its corresponding combination-sensitive neuron in CALM-nepLRB-ANN. Neuromodulation indi-

cates two features based on reward value: (1) serotonergic neuromodulation and (2) dopaminergic neuromodulation. After selecting one output node, the selected output node sends its signal to the corresponding combination-sensitive neuron and it waits until the reward value is received from Observer. In this case, if the reward value is positive, the combination-sensitive neuron releases serotonin; however if the reward is zero, it discharges dopamine. By having these two neurotransmitters, CALM-nepLRB can undo an incorrect behavior which received zero reward; this provides an agent with more opportunities to explore the same context with different behaviors. For example, if an agent executes 'GO_FORWARD' and it acquires zero reward, dopamine is released from the corresponding combination-sensitive neuron and it will directly cause the behavior 'GO_BACKWARD' which is opposite behavior of 'GO_FORWARD'. More details of this are described in Section 5.5.2 and Section 5.5.3.

Recurrent inhibition also occurs based on the pattern of reward values; more precisely, CALM-nepLRB checks the accumulated zero rewards for each output since this context became active. This tells the agent how many times it has failed with each action since it entered its current circumstances. If the number of failures is above a specificed threshold, known as the recurrent threshold, the corresponding behavior is target to be recurrent inhibition. Note that when the agent changes context, the number of failures corresponding to each output node is reset to zero. Also, when the agent gets a positive reward, the count of failures is reset to zero for all output nodes. This is a tool for avoiding infinite repetition of the same behaviors that keep receiving zero reward over time. It should be noted, what is inhibited in not the repetition of the behavior itself. Instead, it is the default learning method of CALM-nepLRB that is inhibited, causing the agent to switch to a backup learning method. The idea is that if the

agent continues to fail, it is likely that its learning method isn't helping it to learn correct behavior in this circumstance, so it should try to learn differently, rather than just try to get out of the circumstance (change behavior) without actually learning a better response.

After performing neuromodulation and checking the recurrent inhibition status, CALM-nepLRB selects an appropriate algorithm between CALM-epLRB (its default learning algorithm) and CALM-rLRB (its backup), which affects the way of setting learning input $LRBX(t)$ and output $LRBY(t)$ for the logistic regression optimization. This mechanism provides an important feature of CALM-nepLRB: theoretically, CALM-nepLRB, in the end, will have received positive reward in every encountered context.

At algorithm Step 7 in Figure 5.6, if there is no recurrent inhibition, the method of setting $LRBX(t)$ and $LRBY(t)$ follows as in CALM-eLRB. Also, at learning step (10), CALM-nepLRB performs Selective-Power-Update, the same as in CALM-epLRB, if it is not in recurrent inhibition status. On the other hand, if an output node is subject to recurrent inhibition, CALM-nepLRB sets the learning input and output in the same way as CALM-rLRB. This is because recurrent inhibition implies that experience-powered learning has not been effective in the current context, therefore it is may be more effective to apply only the current context and reward to the optimization process rather than relying on the past positive experiences. Given this shift from learning based on prior and current experiences to learning only based on the current situation, it should eventually find the rewarding behavior, at least in current context, after optimization. In this way, CALM-nepLRB does not skip zero rewarded context by performing opposite behavior with dopaminergic neuromodulation and selects an appropriate algorithm depending on learning status with recurrent inhibition

biological tools.

In summary, the reasons why CALM-nepLRB is expected to outperform the other algorithms in CALM for many data sets are as follows. First, CALM-nepLRB controls its behavior through neuromodulation of combination-sensitive neurons based on reward value, which gives an agent opportunities to try other behaviors in the same context when the selected behavior is zero rewarded. Second, CALM-nepLRB generates the quasi-target output $R(t)$ in a more sophisticated way with *MASK* so that the ANN can find the appropriate behavior faster and more precisely. Third, CALM-nepLRB has a mechanism, called recurrent inhibition, for avoiding infinite loops of repeating the same behavior with zero reward. Fourth, CALM-nepLRB is able to chooses an appropriate algorithm, either CALM-rLRB or CALM-epLRB, depending on the learning status. Fifth, CALM-nepLRB guarantees that an agent will always find the correct behavior in a given context due to the effects of neuromodulation and recurrent inhibition. This is demonstrated based on the synthetic experimental results in Chapter 6.

5.5.2 CALM-nepLRB-ANN

This section introduces a novel, bio-inspired, generalized, arbitrary-depth, neural network, CALM-nepLRB-ANN, which is designed especially for the CALM-nepLRB algorithm. Figure 5.7 shows CALM-nepLRB-ANN.

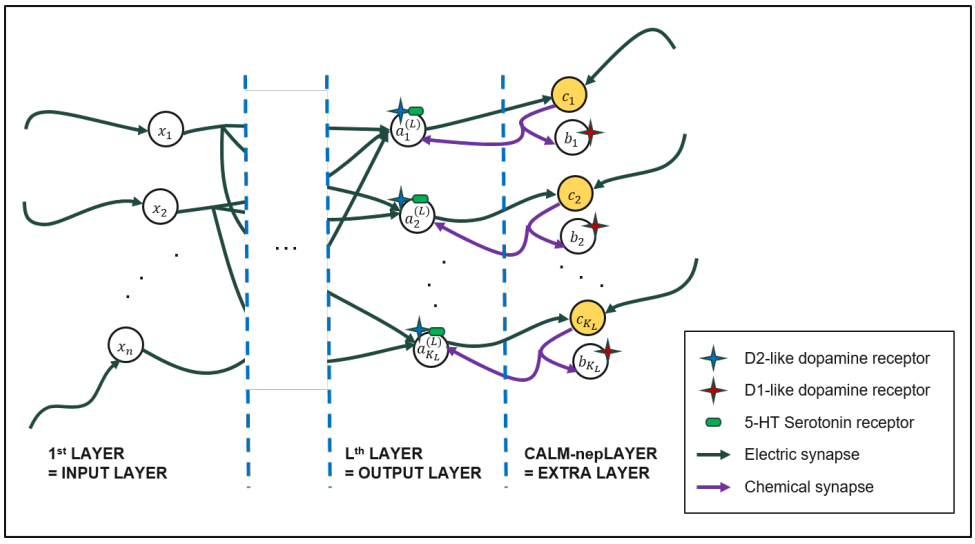


Figure 5.7: CALM-nepLRB-ANN

Compared to a generalized, arbitrary-depth, neural network shown in Figure 3.17, there are five additional novel features in CALM-nepLRB-ANN. First, there is an extra layer named *CALM-nepLayer* which has combination-sensitive neurons and "back neurons". A combination-sensitive neuron is denoted as c_k and a back neurons is represented as b_k where k is between 1 and K_L. Note that it is assumed that a combination-sensitive neuron is activated only when two combined input signals are received where one is from output node and the other one is from Observer with reward. Therefore, at a learning step, only one combination-sensitive neuron is activated since only one output node is selected and the reward value is given to the corresponding combination-sensitive neuron. A *back neuron* refers to an opposite motor neuron of an output node. For example, if a_1 is associated with the 'GO_FORWARD' behavioral task, b_1 is related to 'GO_BACKWARD'.

Second, for each output node, there are three additional neural paths: (1) an electrical synapse from the output node to a combination-sensitive neurons, (2)

a chemical synapse from a combination-sensitive neuron to the output node, and (3) a chemical synapse from a combination-sensitive neuron to a back neuron. Note that each output neuron has a link to a combination-sensitive neuron; and each combination-sensitive neuron has two links; one is connected to the corresponding output node and the other one is linked to a back node.

Third, the chemical synapses of combination-sensitive neurons behave in two different ways based on the reward value form the Observer. If the reward value is positive, a combination-sensitive neuron releases serotonin neurotransmitter; if the reward is zero, it discharges dopamine.

Fourth, each output node has two types of receptor: (1) a 5-HT receptor and (2) a D2-like receptor. If an output node receives serotonin due to a positive reward, it binds to the 5-HT receptor which causes EPSP. On the other hand, if dopamine is released to an output node, it binds to D2-like receptor and causes IPSP.

Fifth, each back neuron has a D1-like receptor which plays a key role in improving the accuracy of CALM-nepLRB. When a combination-sensitive neuron discharges dopamine due to a zero reward, the dopaminergic synapse is activated at the corresponding output node with IPSP and the back neuron node with EPSP. This implies that an agent can undo the incorrect behavior by performing the opposite behavior. For example, if the agent takes an action 'GO_FORWARD' and receives zero reward, the dopaminergic synapse fires the connected back neuron which makes the agent perform 'GO_BACKWARD'. In this case, by performing the opposite behavior via the dopaminergic neuromodulation, a agent can again face the previous context when the previously selected

output was not a good choice.

5.5.3 CALM-nepLRB Learning

In this section, we will see how CALM-nepLRB learns. Algorithm 16 shows the main flow of CALM-nepLRB. Compared to CALM-epLRB-MAIN, in CALM-nepLRB-MAIN, there is one more step at each learning step t for neuromodulation and recurrent inhibition. Note that all the computations steps except for Steps 6, 7, and 8 are same as in CALM-epLRB; thus in this section, we will focus only on the different learning steps: how to generate output, how to set learning input and quasi-target output flexibly, and how to perform neuromodulation and recurrent inhibition.

Algorithm 16 CALM-nepLRB-MAIN Pseudocode

Given $T, ITR, n, L, K_1, \cdots, K_l, \cdots, K_L, \eta, \lambda, \epsilon, \gamma$
for $l = 1$ **to** $L - 1$ **do**
  Init $\theta^{(l)} \in R^{K_{l+1} \times (K_l+1)}$, $\theta^{(l)}(1) \leftarrow \theta^{(l)}$
end for
for $t = 1$ **to** T **do**
  (1) CONTEXT ACQUISITION
  (2) FORWARD PROPAGATION
  (3) OUTPUT SELECTION
  (4) BEHAVIORAL TASK
  (5) REWARD ACQUISITION
  (6) SET UP $R(t) \in \mathbb{R}^{K \times 1}$ based on $r(t)$
  if $r(t) = 1$ **then**
    $\forall_{k \in \{1, K_L\}},\ MASK_k = 1$
    $R_k(t) \leftarrow 1$, if $k = son(t)$
    $R_k(t) \leftarrow 0$, if $k \neq son(t)$
  else if $r(t) = 0$ **then**
    $MASK_{son} = 0$ and $R(t) = MASK$
  end if
  (7) NEUROMODULATION and RECURRENT INHIBITION
  (8) SET UP $LRBX(t)$ AND $LRBY(t)$
  if RECURRENT INHIBITION **then**
    Set as CALM-rLRB Algorithm
    $LRBX(t) \leftarrow context(t)^\top \in \mathbb{R}^{1 \times (n+1)}$
    $LRBY(t) \leftarrow R(t)^\top \in \mathbb{R}^{1 \times K}$
  else
    Set as CALM-epLRB Algorithm
    $m \leftarrow sizeof(EKB, 1)$; number of rows of EKB
    $LRBX(t) \leftarrow [EKB_X;\ context(t)^\top] \in \mathbb{R}^{(m+1) \times (n+1)}$
    $LRBY(t) \leftarrow [EKB_R;\ R(t)^\top] \in \mathbb{R}^{(m+1) \times K}$
  end if
  (9) CALL CALM-LRB-CORE-GEN OPTIMIZATION
  $[J(t),\ \theta(t+1)] \leftarrow$ CALM-LRB-CORE-GEN ($LRBX(t)$, $LRBY(t)$, $\theta(t)$)
  (10) SELECTIVE POWER UPDATE
  if $m > 0$ and NO RECURRENT INHIBITION **then**
    $[\theta(t+1)] =$ CALM-SELECTIVE-POWER-LEARNING($\theta(t+1)$, EKB)
  end if
  (11) SAVE CURRENT EXPERIENCE INTO EKB
  if $r(t) = 1$ **then**
    $EKB \leftarrow add(context(t), son(t), r(t), selected\_weight(t))$
  end if
end for

• Quasi-target output vector $R(t)$ and $MASK$ are generated based on reward value $r(t)$ as follows. Note that the size of $R(t)$, $MASK$, and $a^{(L)}$ are the same as K_L.

$$MASK = R(t) = \begin{bmatrix} R_1(t) \\ R_2(t) \\ \vdots \\ R_k(t) \\ \vdots \\ R_{K_L}(t) \end{bmatrix}_{K_L \times 1}$$

If $(r(t) = 1)$

$\forall_{k \in \{1, K_L\}},\ MASK_k = 1$

$R_k(t) \leftarrow 1$ if $k = son(t)$ and $r(t) = 1$

$R_k(t) \leftarrow 0$ if $k \neq son(t)$ and $r(t) = 1$

$R_k(t) \leftarrow 0$ if $k = son(t)$ and $r(t) = 0$

$R_k(t) \leftarrow 1$ if $k \neq son(t)$ and $r(t) = 0$

If $(r(t) = 0)$

$MASK_{son} = 0$

$R_k(t) = MASK$

• Neuromodulation is processed in two paths depending on the reward value as covered in CALM-nepLRB-ANN in Section 5.5.2. In the computational process, this neuromodulation is mostly related to setting target values as described above. After the reward acquisition, only the combination-sensitive neuron corresponding to the selected output nodes is activated with the reward signal and

then it releases serotonin when there is positive reward and dopamine when there is zero reward. In this case, the neural pathways of serotonin and dopamine from the combination-sensitive neuron toward the output node are computationally covered by the quasi-target output $R_{son}(t)$. For example, in the case where the value of the selected output node is 1, if the reward is zero, $R_{son}(t)$ would be zero and this serves as the IPSP effect of the dopamine with the D2-like receptor on the output node; or, if reward is positive, $R_{son}(t)$ is set to be 1 as described above, and this serves as the EPSP effects of serotonin with binding 5-HT receptor on the output node. On the other hand, the dopaminergic pathway towards the back node is activated by performing the opposite behavior of the original behavior corresponding to the selected output node. In this way, the processes involving two types of neuromodulation of serotonin and dopamine is embodied with setting the quasi-target output $R(t)$, which supports a bio-inspired learning mechanism.

• Recurrent inhibition uses the simple calculation of counting the number of accumulated zero rewards for each output node since it started to receive zero rewards. If the number of accumulated zero rewards is over the recurrent threshold, the learning status is one of recurrent inhibition so that an agent can avoid infinite repetition of incorrect behavior. The count of these accumulated zero rewards is reset to zero when the agent receives positive reward.

• Learning input $LRBX(t)$ and learning output $LRBY(t)$ is generated as follows based on context and reward value. Note that when recurrent inhibition is necessary due to the number of incorrect behavior is exceeding the recurrent threshold, CALM-nepLRB sets the learning input and output in the same way as in CALM-rLRB; otherwise it uses the same method as CALM-eLRB with

additional Selective-Power-Update, which is same as for CALM-epLRB.

If (recurrent inhibition)

$LRBX = context(t)^\top$ $\in \mathbb{R}^{1\times(n+1)}$

$LRBY = R(t)^\top$ $\in \mathbb{R}^{1\times K_L}$

If (no recurrent inhibition)

$m \leftarrow sizeof(EKB, 1)$; number of rows of EKB

$EKB_X \in \mathbb{R}^{(m)\times(n+1)}$

$EKB_R \in \mathbb{R}^{(m)\times K}$

$LRBX(t) = [EKB_X;\ context(t)^\top]$ $\rightarrow LRBX(t) \in \mathbb{R}^{(m+1)\times(n+1)}$

$LRBY(t) = [EKB_R;\ R(t)^\top]$ $\rightarrow LRBY(t) \in \mathbb{R}^{(m+1)\times K}$

Chapter 6

CALM Experiments and Results

This chapter describes the experiments designed for evaluating CALM algorithms and shows the experimental results. In this dissertation, five synthetic data sets are generated to give CALM different environmental complexities and on each data set four different CALM learning algorithms are evaluated with five depths of CALM-ANN to check its performance on different neural complexities. Section 6.1 explains experimental setup and evaluation methods. Section 6.2 shows experimental results of four different CALM algorithms where each of which runs with five depths of CALM-ANN on five different synthetic data sets.

6.1 Experimental Setup

This section explains experimental designs including neural network topologies, learning parameter values, synthetic data sets, and evaluation methods.

6.1.1 CALM-ANNs

In the experiments of this dissertation, five different depths of CALM-ANN are used in evaluating CALM, where each ANN has different number of neural layers: CALM-ANN1 ($L = 2$), CALM-ANN2 ($L = 3$), CALM-ANN3 ($L = 4$), CALM-ANN4 ($L = 5$), and CALM-ANN5 ($L = 6$). CALM-ANN1 has only one layer of weights connecting inputs to the output nodes and thus it has no hidden layer. Recall that, in this dissertation, the number of layers of an ANN

refers to the total number of layers including input, output, and hidden layers as described in Section 1. Therefore CALM-ANN1 has two number of layers in total and thus the learning parameter L is set to be 2 when an CALM algorithm uses it. CALM-ANN2 has two layers of weights and it has one hidden layer. CALM-ANN3 has three layers of weights and it has two hidden layers. CALM-ANN4 has four layers of weights and it has three hidden layers. CALM-ANN5 has five layers of weights and it has four hidden layers.

In each CALM-ANN, the number of input nodes is 3 including bias nodes ($n = 2,\ n+1 = 3$) and the number of output nodes at the last layer is 7 ($K_L = 7$). The total number of each hidden layer is 26 as each hidden layer includes a bias node ($K_l = 26$ where $l \in (1, L)$); however, the number of net nodes in each hidden layer is 25 ($K_l = 25$ where $l \in (1, L)$) since there are two types of node in each hidden layer of an ANN in this dissertation as shown in Figure 3.17 in Section 3.2. Note that l is in open interval $(1, L)$, not close interval $[1, L]$, which excludes input and output layer. For example, CALM-ANN5 has 3 input nodes, 7 actual output nodes, and 25 hidden net and 26 actual output nodes at each second, third, fourth, and fifth hidden layer.

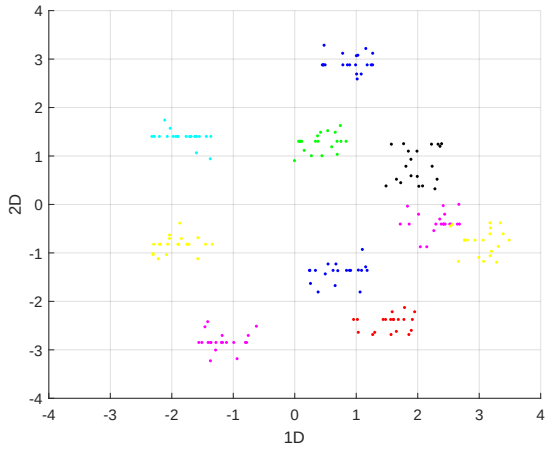
Each CALM-ANN has different initial weight values as each one has different number of layers of weights. CALM-ANN1 has only one layer of weights, so the number of weights are $3 \times 7 = 21$ and they are randomly distributed between -0.1 and $+0.1$. CALM-ANN2 has two layers of weights, so the number of weights are $3 \times 25 + 26 \times 7 = 257$ and they are randomly distributed between -0.1 and $+0.1$. CALM-ANN3 has three layers of weights, so the number of weights are $3 \times 25 + 26 \times 25 + 26 \times 7 = 907$ and they are randomly distributed between -0.1 and $+0.1$. CALM-ANN4 has four layers of weights, so the number of weights are $3 \times 25 + 26 \times 25 + 26 \times 25 + 26 \times 7 = 1,557$ and they are randomly

distributed between -0.1 and $+0.1$. CALM-ANN5 has five layers of weights, so the number of weights are $3 \times 25 + 26 \times 25 + 26 \times 25 + 26 \times 25 + 26 \times 7 = 2,207$ and they are randomly distributed between -0.1 and $+0.1$.

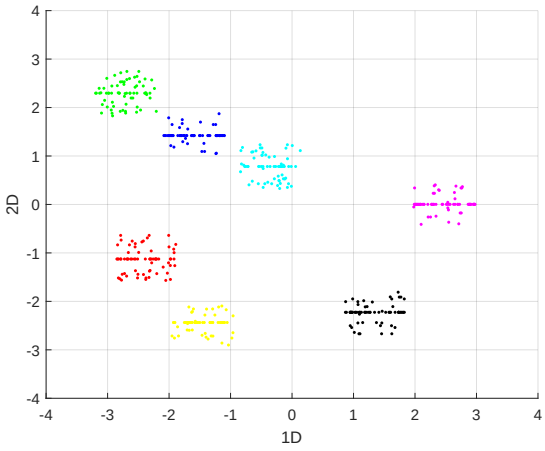
In each experiment on each different data set with each different CALM learning algorithm, all CALM-ANNs used with the same initial weight distribution as described above so as to compare its performances under the same condition of initial weights.

6.1.2 Synthetic Data Sets

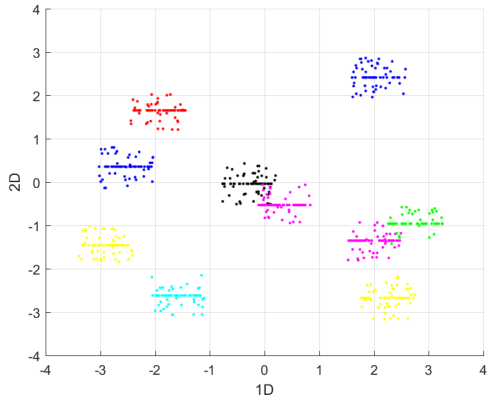
There are five synthetic data sets used in the experiments which are shown as Figure 6.1: DATA1, DATA2, DATA3, DATA4, and DATA5. Each data set has different form of data distribution. Having different synthetic data set is for evaluating CALM algorithms in different input complexities in a general way before applying them to real domain. Synthetic data means that it includes virtual input and virtual target output for each virtual input. Each data point in a data set represents both virtual input and corresponding virtual target output which will be used for checking reward value. Virtual input data is encoded with the 2D Cartesian value and the virtual target output is represented with different color. As the number of input nodes are 3 and the number of output nodes at the last layer in each CALM-ANN is 7, the dimensionality of each data set is 2D which will be 3D after including bias features in CALM learning process. The 7 target outputs are represented with each different corresponding color: green-OUTPUT1, blue-OUTPUT2, yellow-OUTPUT3, magenta-OUTPUT4, cyan-OUTPUT5, black-OUTPUT6, red-OUTPUT7.



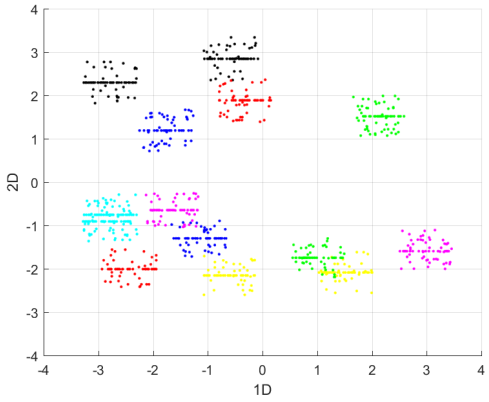
(a) Data1



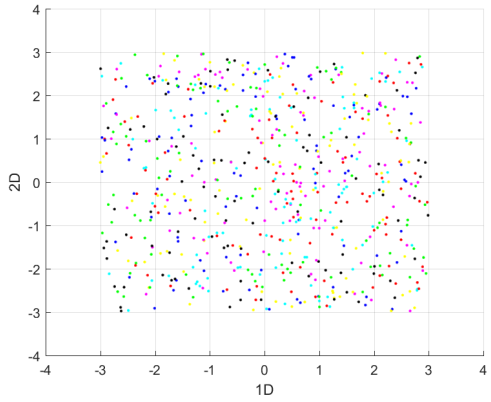
(b) Data2



(c) Data3



(d) Data4



(e) Data5

Figure 6.1: Synthetic Data Sets

In terms of feeding a synthetic data set into CALM learning process, each

data point is assumed as sensory data. Specifically, in each discrete learning step t, the Sensory System iteratively takes a data point as an input data example and Context Supplier gets the data point as a context which is in the form of Cartesian value; and then this context is directly feed into a CALM-ANN. Therefore, the number of data points in a synthetic data set signifies the maximum discrete learning step T. Note that all the synthetic data sets in this dissertation are randomly shuffled. This means Context Supplier does not get a context cluster by cluster, but gets in a random way. Specifically, a synthetic data is not sequentially ordered by virtual target output so that it can support unpredictable environment.

In terms of getting reward values based on a synthetic data set, in each learning step, the Observer will check if a selected output node from a CALM-ANN is same as the given virtual target output of the data point. If the selected output node is same as the given virtual target output, Observer gets reward $r(t)$ as 1; or if it is not same, $r(t)$ is set to be 0. In this way, CALM is able to simulate its reward-based learning algorithms based on the synthetic data sets.

DATA1 includes 10 data clusters where each cluster has 20 data points thus the total number of data points in DATA1 are 200. Each cluster represents one of possible target outputs; we can see DATA1 has 2 clusters for OUTPUT3, 2 clusters for OUTPUT2, 2 clusters for OUTPUT7, and one cluster for the rest. DATA2 includes 7 data clusters each with 70 data points thus the total number of data points are 490. Likewise, each cluster represents one of possible target outputs thus DATA2 has one cluster for each of the 7 outputs. DATA3 includes 10 data clusters each with 70 data points thus the total number of data points are 700. DATA3 has two clusters for OUTPUT2, two clusters for OUTPUT3,

two clusters for OUTPUT4, and one cluster for the rest. DATA4 includes 14 clusters each with 70 data points thus the total number of data points are 980. DATA4 has exactly 2 clusters for each target output. DATA5 has 700 randomly generated data points between -3 and 3 where there is no data clusters. Each data point is targeting one of seven possible target output randomly.

In generating and understanding the above data sets, there are four different factors which affects learning performance: (1) number of data points, (2) number of clusters, (3) the level of overlap areas of clusters, and (4) distribution of clustered target outputs. DATA1 and DATA3 each has 10 clusters but the number of data points per cluster is different; DATA1 has 20 data points per cluster while DATA3 has 70 data points per cluster. Also, DATA1 has only one overlapped data cluster while DATA3 has two overlapped data clusters. DATA2 has 7 clusters, exactly one for each different target output, and there are no overlapped clusters. DATA4 has 2 clusters, exactly one for each output, and it has more overlapped data clusters points compared to DATA1 and DATA3. DATA5 is highly non-structured, non-linear, complex data set and it is to evaluate how each CALM algorithm adapts to unpredictable environment. Based on this synthetic data sets, the experimental results are described in Section 6.2.

6.1.3 Evaluation Methods

In order to increase the reliability of experimental results, each algorithm of CALM is evaluated by 5-fold cross-validation with the ratio of 80% of training and 20% of testing data from a synthetic data set. Training data refers to the data which is given to CALM learning process and testing data is the data which

is used only to evaluate its performance by using the learned CALM-ANNs. In 5-fold cross-validation, a data set is re-represented by five folds and each fold contains different combination of testing and training data. Note that in this dissertation each different combination is denoted as a fold. Table 6.1 shows how a data set can be represented with five folds. In each fold, only training data is used to train CALM-ANNs and testing data is used to evaluate the learning model with the trained neural weights as a learning results; in this way, we can also compare the learning results of training and testing data.

Sub Sets	Fold1	Fold2	Fold3	Fold4	Fold5
20%DATA	Testing	Training	Training	Training	Training
20%DATA	Training	Testing	Training	Training	Training
20%DATA	Training	Training	Testing	Training	Training
20%DATA	Training	Training	Training	Testing	Training
20%DATA	Training	Training	Training	Training	Testing

Table 6.1: 5-Fold Cross Validation for Each Data Set

In extracting testing data from a synthetic data set for generating a fold, a data set is simply divided into five different data subsets and each of them is used for testing data for each fold. Recall that a synthetic data set itself is randomly shuffled data as described in Section 6.1.2. In this case, each testing data in each fold has different set of randomly selected data points from a synthetic data set; and of course all of the testing data sets from all folds is same as the synthetic data set. In other words, each testing data set does not include same number of data points from each cluster. For example, the number of training data points in DATA1 is 160 and the number of testing data points is 40 which is 20% of DATA1. In this case, the 40 data points in fold1 are randomly selected from the entire set of 200 data points. Likewise,

40 data points in fold2 are also randomly selected but among 160 data points which exclude the 40 data points in fold1. Likewise, each training data in each fold is also randomly selected data as the corresponding synthetic data set itself is randomly shuffled; this implies that Context Supplier takes each context in a random way, not cluster by cluster, which mimics unpredictable environment.

On each fold from each synthetic data set, four different CALM algorithms are evaluated with the five different depths of CALM-ANN. The performance of each algorithm of CALM is evaluated in five types of measurement: (1) accuracy, (2) cost function, (3) accumulated rewards, (4) dynamic accuracy, and (5) dynamic EKB transition. The learning parameters are set: $\eta = 1.0,\ \lambda = 0.0,\ \epsilon = 0.3,\ \gamma = 0.5$, and $ITR \in \{1, 200\}$. Note that only the number of iterations, ITR, has two different experimental values and the others has no variance in this dissertation.

6.2 Experimental Results on Synthetic Data Sets

This section shows experimental results on the five evaluation methods: (1) accuracy, (2) cost function, (3) accumulated rewards, (4) dynamic accuracy, and (5) dynamic EKB transition. Based on five synthetic data sets, four different CALM learning algorithms with five depths of CALM-ANN are evaluated and the results are described in each sub evaluation section.

6.2.1 Accuracy Analysis

The first measurement of CALM performance is accuracy. Accuracy is for testing how a learning algorithm can correctly classify given contexts through the learning process. In accuracy analysis, there are two types of learning result where one is from training data and the other one is from testing data. Training accuracy is measured in each learning step by calculating percentage of the number of successfully classified contexts in a incremental way. Note that over the learning steps, the number of processed contexts increases since CALM iteratively takes one context at each learning step. For example, if current learning step is 10 ($t = 10$), this means CALM processed 10 contexts and training accuracy at this learning step is checking how many of the past 10 contexts the current CALM-ANN can correctly classify; therefore if current learned CALM-ANN at learning step 10 can successfully classify 8 past contexts, the accuracy is 80% at that learning step. On the other hand, the testing accuracy is acquired by applying current CALM-ANN, which learned from the 10 contexts of training data, into the whole testing data in each learning step. For example, on fold1 of DATA1, if current learning step is 10, testing accuracy is acquired by applying current CALM-ANN into the whole testing data which is 40 context points for DATA1 while the training accuracy is acquired by applying current CALM-ANN into the 10 passed data; In this way, accuracy analysis gives us an indicator of the level of precise context-awareness over the learning steps by comparing the results from training and testing data.

Figure 6.2 and Figure 6.3 shows the training and testing accuracy of all CALM algorithms with five depths of CALM-ANN on fold1 of DATA1 through the incremental learning step. In the figures, left columns are when the number

of iteration is 1 ($ITR = 1$) and the right columns are when it is 200 ($ITR = 200$). Each data point in Figure 6.2 refers to a percentage of a learning algorithm correctly classifying the given incremental input contexts from training data. On the other hand, each data point in Figure 6.3 means the ability of a learning algorithm classifying a fixed number of testing data after learning the currently given incremental input contexts from the training data. Likewise, Figure 6.4 and Figure 6.5 are the accuracy of all CALM algorithms on both fold1 training and testing data of DATA2. Figure 6.6 and Figure 6.7 are the accuracy of all CALM algorithms on both fold1 training and testing data of DATA3. Figure 6.8 and Figure 6.9 are the accuracy of all CALM algorithms on both fold1 training and testing data of DATA4. Figure 6.10 and Figure 6.11 are the accuracy of all CALM algorithms on both fold1 training and testing data of DATA5.

Table 6.2 shows both training and testing accuracy of all of CALM algorithms with five depths of CALM-ANN on each fold of DATA1; specifically, the training accuracy on the table is when the number of iterations is 200 ($ITR = 200$) at the end of learning step ($t = 160$) and the testing accuracy is when the number of iterations is 200 and the learning step is 40. Likewise, Table 6.3, Table 6.4, Table 6.5, and Table 6.6 shows both training and testing accuracy of all of CALM algorithms with five different depths of CALM-ANN on each fold of DATA2, DATA3, DATA4, and DATA5 respectively when $ITR = 200$ and t is at the end of learning step.

		Fold1		Fold2		Fold3		Fold4		Fold5	
CALM-	L	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te
rLRB	2	9.4	12.5	10.6	20	16.9	22.5	10.6	7.5	19.4	22.5
	3	9.4	5	43.1	30	33.8	17.5	11.9	17.5	35.6	25
	4	36.9	40	20	20	10.6	7.5	16.9	5	9.4	12.5
	5	9.4	12.5	20	20	17.5	30	10.6	7.5	8.8	15
	6	10	10	11.3	5	11.3	5	8.1	17.5	10	10
eLRB	2	40.6	42.5	33.1	17.5	28.1	37.5	30.6	27.5	30.6	27.5
	3	67.5	57.5	60	50	79.4	77.5	77.5	82.5	71.3	62.5
	4	68.8	65	20	20	20.6	17.5	18.8	25	20	20
	5	48.8	35	70.6	60	71.3	62.5	73.8	65	77.5	80
	6	21.3	15	58.8	32.5	50	50	48.8	55	57.5	70
epLRB	2	40	40	38.1	22.5	28.1	37.5	30.6	27.5	31.3	27.5
	3	65.6	65	45	20	66.3	52.5	68.1	67.5	62.5	55
	4	67.5	70	20	20	20.6	17.5	18.8	25	20	20
	5	68.1	72.5	69.4	70	74.4	65	70	70	66.9	72.5
	6	21.3	15	46.9	20	48.8	37.5	37.5	37.5	38.8	45
nepLRB	2	55	47.5	63.1	42.5	54.4	40	50	62.5	51.9	62.5
	3	99.4	97.5	100	95	98.1	97.5	98.1	100	98.1	97.5
	4	99.4	97.5	100	95	99.4	100	99.4	100	100	97.5
	5	98.8	97.5	100	95	53.1	37.5	35.6	57.5	51.3	45
	6	98.8	100	100	90	98.1	97.5	97.5	100	98.8	100

Table 6.2: CALM Accuracy (%) on Data 1 ($ITR = 200$)

		Fold1		Fold2		Fold3		Fold4		Fold5	
CALM-	L	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te
rLRB	2	16.1	18.4	43.6	39.8	14	14.3	18.1	13.3	22.2	18.4
	3	76.8	70.4	58.4	54.1	65.6	66.3	67.9	67.3	77.3	74.5
	4	29.6	24.5	15.6	9.2	15.1	11.2	13.5	17.3	14.5	14.3
	5	15.1	11.2	14	15.3	12.5	21.4	0	0	14.8	12.2
	6	15.1	11.2	15.6	9.2	17.3	23.5	15.1	11.2	14.8	12.2
eLRB	2	42.1	45.9	40.8	51	44.6	35.7	45.2	33.7	42.3	44.9
	3	100	100	100	100	100	100	100	100	100	100
	4	85.7	85.7	99.7	100	100	100	86.2	82.7	84.9	87.8
	5	100	100	98.7	99	58.4	52	69.9	77.6	72.4	67.3
	6	85.2	85.7	85.7	83.7	84.9	82.7	84.4	87.8	73.2	67.3
epLRB	2	42.1	45.9	40.8	51	44.6	35.7	45.2	33.7	42.3	44.9
	3	100	100	100	100	100	100	100	100	100	100
	4	85.7	85.7	99.7	100	87.5	78.6	86.5	82.7	85.2	87.8
	5	100	100	100	100	100	100	100	100	100	100
	6	43.1	41.8	86	84.7	86	84.7	84.7	87.8	85.2	85.7
nepLRB	2	99.5	99	100	100	100	100	99.7	100	96.7	93.9
	3	100	100	100	100	100	100	100	100	100	100
	4	100	100	100	100	100	100	100	100	100	100
	5	100	100	100	100	100	100	100	100	100	100
	6	100	100	100	100	100	100	100	100	100	100

Table 6.3: CALM Accuracy (%) on Data 2 ($ITR = 200$)

		Fold1		Fold2		Fold3		Fold4		Fold5	
CALM-	L	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te
rLRB	2	19.1	20.7	20.4	18.6	9.6	11.4	11.8	15.7	32.1	21.4
	3	41.3	45.7	67	69.3	49.3	48.6	40.5	37.1	43.6	40.7
	4	13.4	15.7	20.4	18.6	14.8	13.6	10	10	9.8	10.7
	5	9.5	12.1	10.5	7.9	9.6	11.4	10	10	10.4	8.6
	6	10.4	8.6	10.4	8.6	10.4	8.6	10	10	21.4	14.3
eLRB	2	51.6	43.6	39.6	41.4	39.1	43.6	40.9	36.4	41.1	35.7
	3	97.9	95.7	97.9	98.6	97.3	95	98	96.4	88.9	91.4
	4	58.9	64.3	97.9	97.9	96.6	99.3	97.5	95	96.6	99.3
	5	97.3	92.9	87	84.3	68.9	61.4	68.4	67.9	61.1	47.9
	6	78.8	75.7	90.2	93.6	95.5	96.4	85.9	92.1	87.7	84.3
epLRB	2	51.6	43.6	49.5	52.1	51.6	43.6	49.6	51.4	51.3	45
	3	97	93.6	95.7	95.7	97	97.1	97.3	97.9	96.4	99.3
	4	86.3	88.6	87	90	96.4	95	97.9	97.9	97	100
	5	87	87.9	78.9	76.4	77.3	74.3	87.1	88.6	88	87.1
	6	67	71.4	96.8	97.1	76.6	75.7	76.4	77.1	78.2	69.3
nepLRB	2	76.8	75	57.7	51.4	30.2	37.9	87.3	82.1	84.8	82.1
	3	99.3	95	98.4	97.1	98.4	97.9	98.2	97.9	97.5	98.6
	4	99.8	94.3	99.3	96.4	98.9	97.1	98.9	97.9	97.7	98.6
	5	99.8	93.6	99.5	98.6	99.5	96.4	99.1	96.4	99.6	100
	6	99.8	94.3	99.5	97.9	99.8	94.3	100	96.4	99.1	100

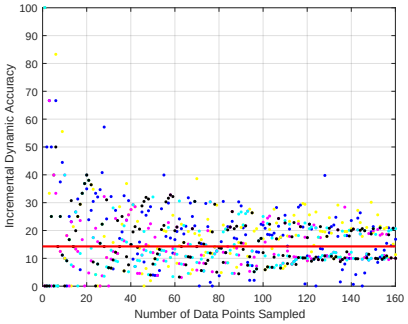
Table 6.4: CALM Accuracy (%) on Data 3 ($ITR = 200$)

		Fold1		Fold2		Fold3		Fold4		Fold5	
CALM-	L	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te
rLRB	2	22.4	11.7	13.1	10.7	7.9	4.1	20.7	22.4	28.4	25
	3	11.5	6.1	22.7	16.8	27.2	33.7	13.5	17.3	9.1	8.2
	4	20.7	16.3	21.3	21.4	26.5	32.1	17.5	16.8	9.2	10.2
	5	0	0	21.9	19.4	13.6	16.8	16.8	17.9	14.3	14.3
	6	14.5	13.3	15.4	9.7	15.7	8.7	13.5	17.3	17.6	20.4
eLRB	2	34.8	38.8	29.6	24.5	30.9	19.4	27.9	31.1	28.8	27.6
	3	81.3	76	77.8	78.1	84.7	79.6	84.6	86.2	84.8	82.7
	4	85.1	82.1	85.3	83.2	95.9	98	95.9	94.4	96.8	96.4
	5	69.8	73.5	89	92.9	79.6	78.6	70.4	64.8	69.6	70.4
	6	69.1	68.9	57.4	55.6	60.7	51.5	56.4	57.7	43.2	41.3
epLRB	2	34.7	38.8	36	34.7	36.7	31.6	35.2	37.8	36.1	34.2
	3	73.3	67.3	78.1	79.1	70.8	74	78.7	77	79.1	75.5
	4	91.5	88.3	88.9	88.3	83.5	82.1	90.6	86.7	90.4	88.8
	5	77.3	73.5	34.3	30.1	57.7	54.1	43	41.8	50.5	48
	6	80.6	84.7	44.8	38.8	37.4	29.1	35.5	36.7	35.6	36.2
nepLRB	2	50.6	51.5	37.2	35.2	56.6	49	37.2	38.8	57.3	52
	3	98.9	96.4	98.2	98	98.3	97.4	98.3	96.9	98	95.9
	4	98.9	96.4	98.7	96.4	98.7	96.9	98.7	96.9	98.5	96.4
	5	99.2	96.4	99.1	97.4	98.9	97.4	99.1	98	99	96.4
	6	99.5	95.9	98.5	98	98.6	97.4	98.9	98.5	99.2	96.4

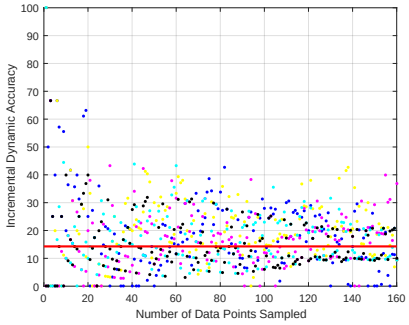
Table 6.5: CALM Accuracy (%) on Data 4 ($ITR = 200$)

		Fold1		Fold2		Fold3		Fold4		Fold5	
CALM-	L	Tr	Te	Tr	Te	Tr	Te	Tr	Te	Tr	Te
rLRB	2	14.8	17.1	15.2	12.1	15.2	10.7	14.1	13.6	16.3	15
	3	16.1	19.3	11.6	15	14.3	11.4	14.3	13.6	15.5	20
	4	14.1	16.4	14.3	14.3	14.3	14.3	14.3	14.3	14.3	14.3
	5	14.3	14.3	16.8	12.9	14.3	14.3	14.3	14.3	13.2	14.3
	6	14.3	14.3	14.3	14.3	14.3	14.3	14.3	14.3	14.3	14.3
eLRB	2	16.3	15	15.5	16.4	15.7	17.9	16.1	14.3	15.9	17.1
	3	20.7	12.9	18.6	13.6	18.8	12.1	18.6	9.3	17.1	15.7
	4	20.7	15	17.9	8.6	17.9	20	17.5	16.4	17.7	8.6
	5	16.1	11.4	18	13.6	18.4	20	18	16.4	18	15
	6	17.9	13.6	16.8	13.6	16.6	15.7	16.4	16.4	17.3	15.7
epLRB	2	16.3	15	15.5	16.4	15.7	17.9	16.1	14.3	15.9	17.1
	3	20.7	10.7	17.7	11.4	17.5	16.4	18.4	15	17.5	15.7
	4	20.2	13.6	17.7	12.1	17.1	13.6	18	18.6	20.9	15
	5	14.5	10.7	17	14.3	17.3	14.3	17.5	15	17	15
	6	14.8	12.9	18	15	16.4	17.1	17.5	17.9	16.4	17.1
nepLRB	2	18.4	16.4	19.1	14.3	18.9	17.1	17.9	17.9	18	21.4
	3	22.7	17.1	15.4	14.3	26.6	10	17.5	15	24.5	20
	4	22.5	12.1	32.5	12.9	29.6	13.6	23.6	15	17.3	15
	5	14.8	13.6	20	12.9	16.3	14.3	20.7	12.1	22.7	12.1
	6	23.6	12.9	14.3	14.3	19.6	13.6	23.4	15.7	18.6	14.3

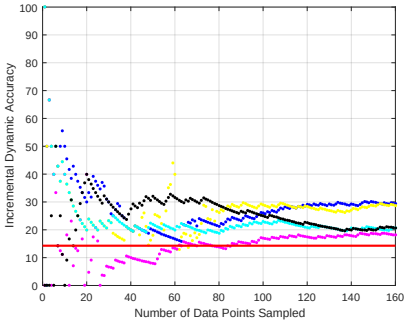
Table 6.6: CALM Accuracy (%) on Data 5 ($ITR = 200$)



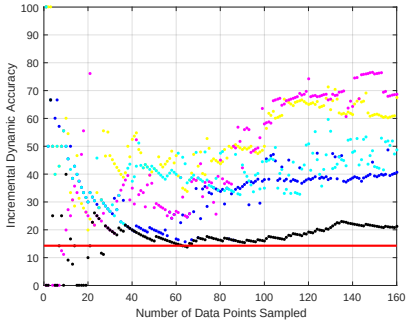
(a) CALM-rLRB ($ITR = 1$)



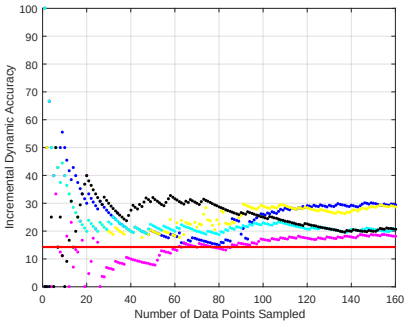
(b) CALM-rLRB ($ITR = 200$)



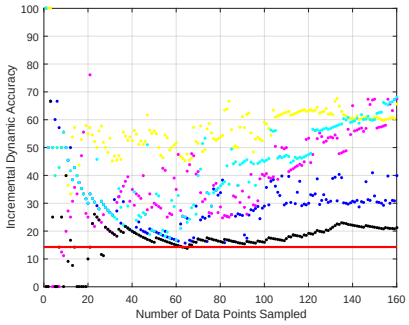
(c) CALM-eLRB ($ITR = 1$)



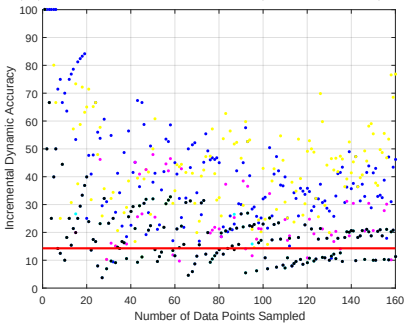
(d) CALM-eLRB ($ITR = 200$)



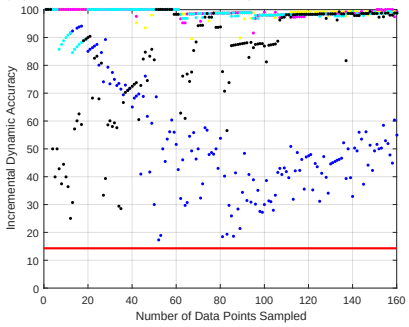
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

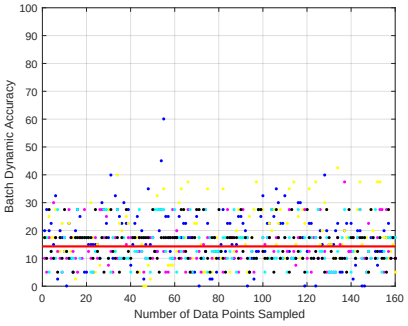


(g) CALM-nepLRB ($ITR = 1$)

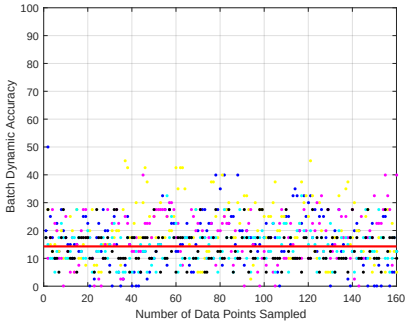


(h) CALM-nepLRB ($ITR = 200$)

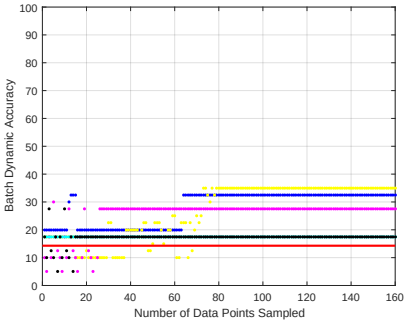
Figure 6.2: Accuracy on Data 1 - Training Fold 1



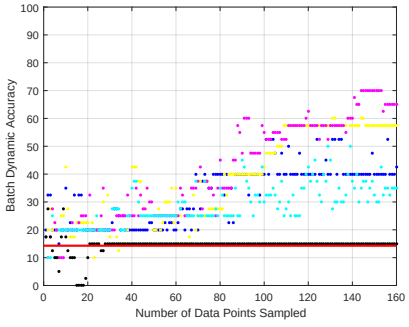
(a) CALM-rLRB ($ITR = 1$)



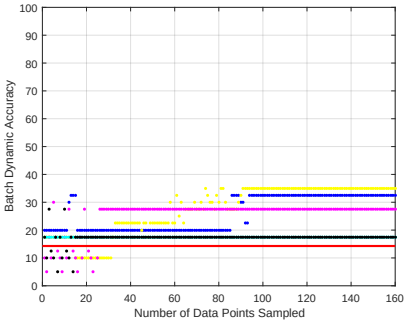
(b) CALM-rLRB ($ITR = 200$)



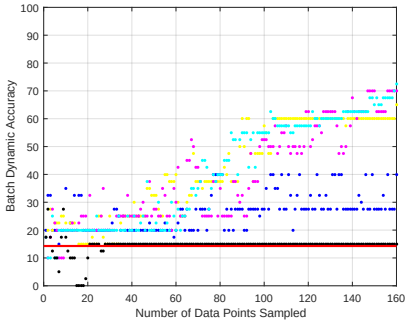
(c) CALM-eLRB ($ITR = 1$)



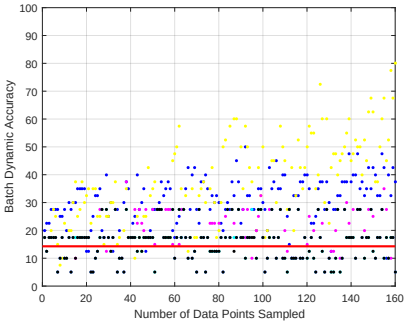
(d) CALM-eLRB ($ITR = 200$)



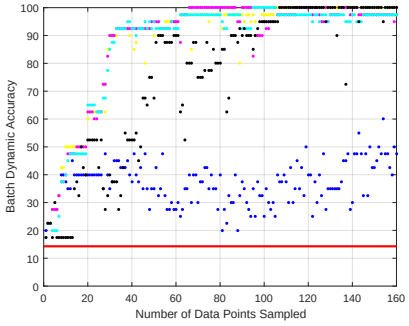
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

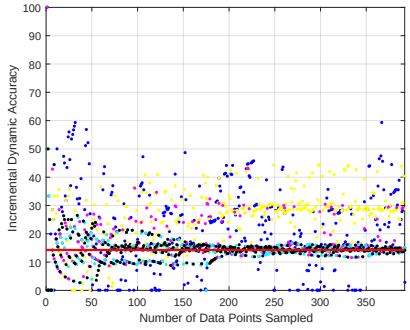


(g) CALM-nepLRB ($ITR = 1$)

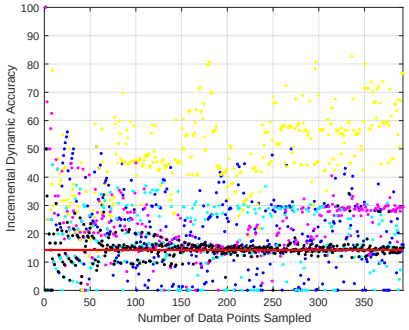


(h) CALM-nepLRB ($ITR = 200$)

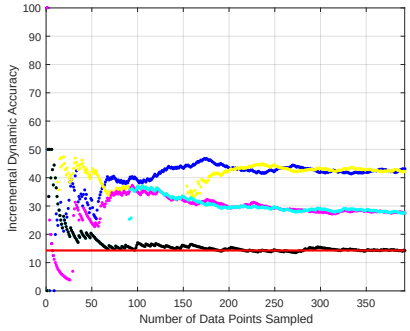
Figure 6.3: Accuracy on Data 1 - Testing Fold 1



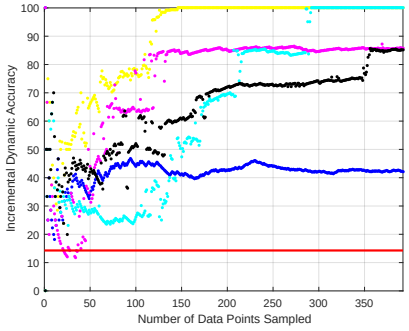
(a) CALM-rLRB ($ITR = 1$)



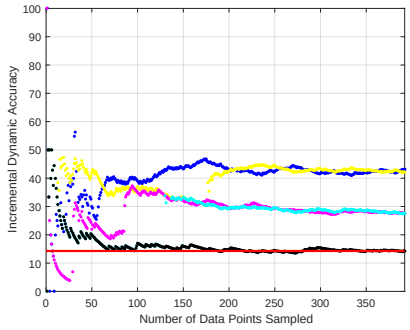
(b) CALM-rLRB ($ITR = 200$)



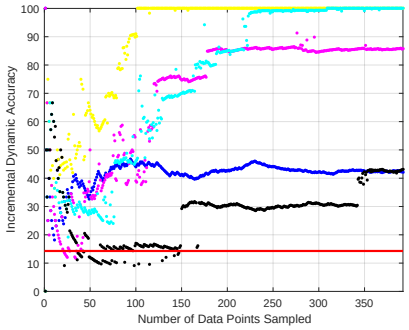
(c) CALM-eLRB ($ITR = 1$)



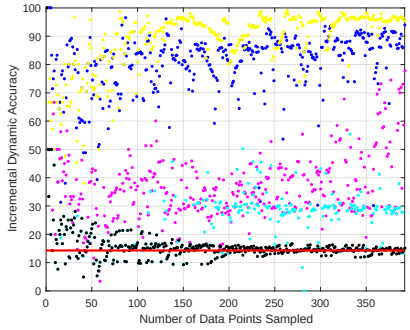
(d) CALM-eLRB ($ITR = 200$)



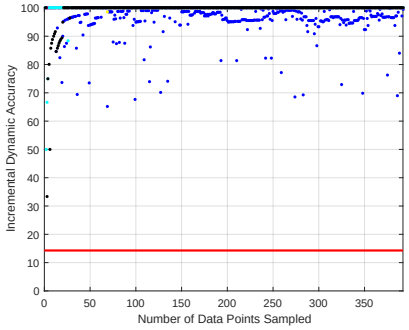
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

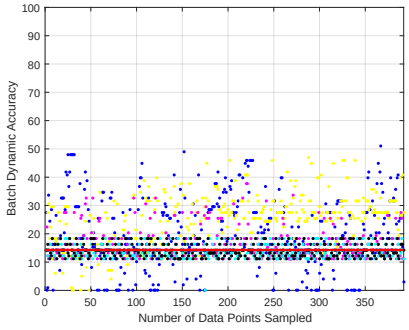


(g) CALM-nepLRB ($ITR = 1$)

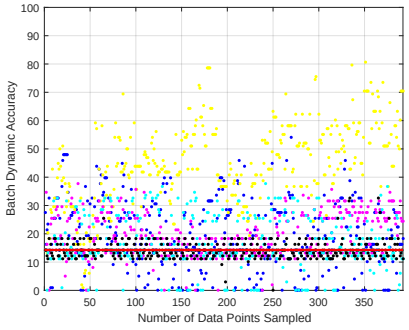


(h) CALM-nepLRB ($ITR = 200$)

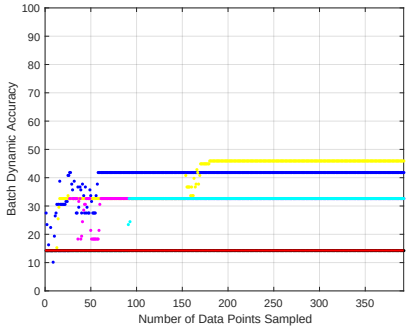
Figure 6.4: Accuracy on Data 2 - Training Fold 1



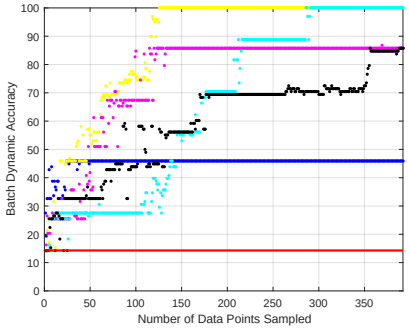
(a) CALM-rLRB ($ITR = 1$)



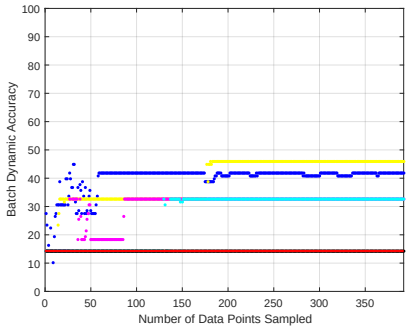
(b) CALM-rLRB ($ITR = 200$)



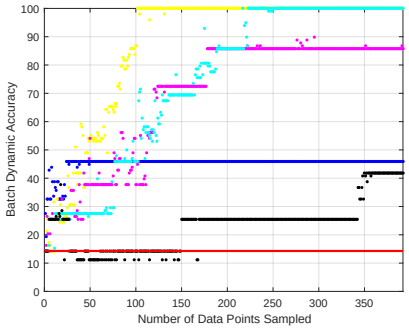
(c) CALM-eLRB ($ITR = 1$)



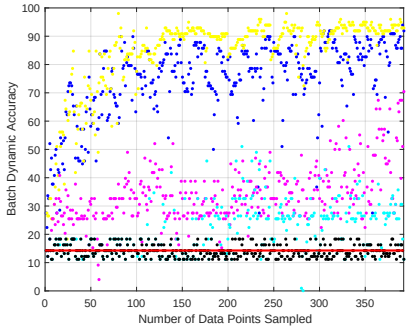
(d) CALM-eLRB ($ITR = 200$)



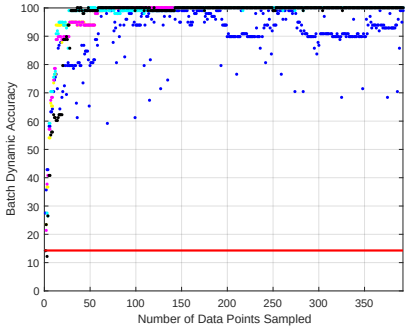
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

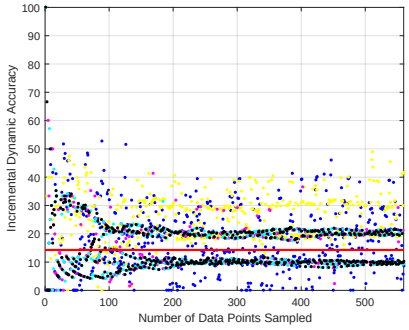


(g) CALM-nepLRB ($ITR = 1$)

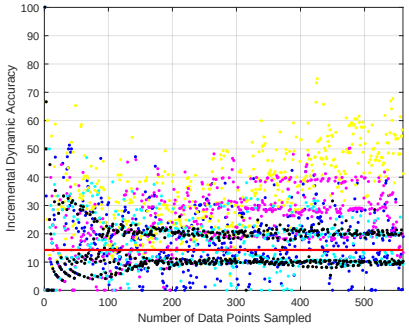


(h) CALM-nepLRB ($ITR = 200$)

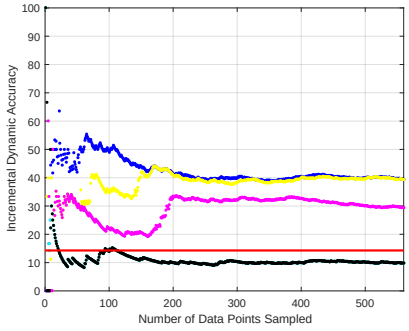
Figure 6.5: Accuracy on Data 2 - Testing Fold 1



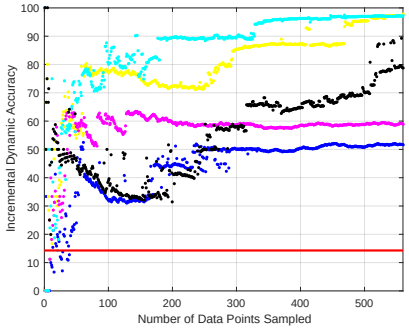
(a) CALM-rLRB ($ITR = 1$)



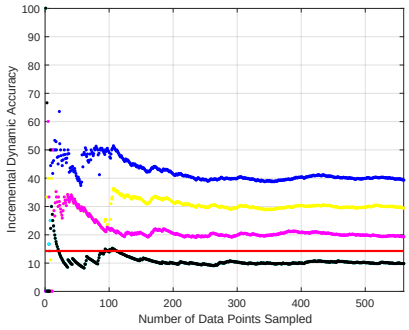
(b) CALM-rLRB ($ITR = 200$)



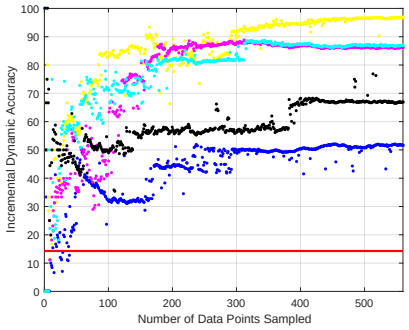
(c) CALM-eLRB ($ITR = 1$)



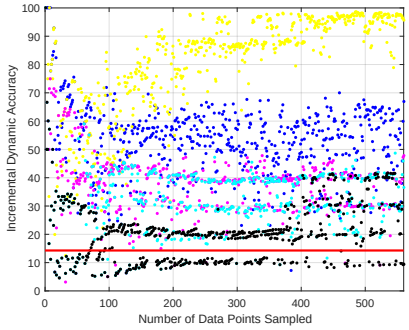
(d) CALM-eLRB ($ITR = 200$)



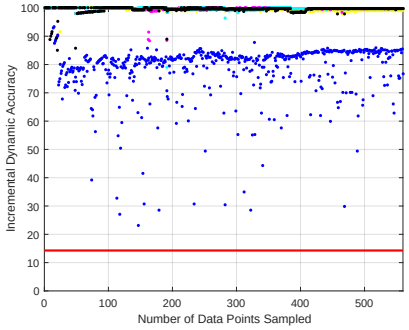
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

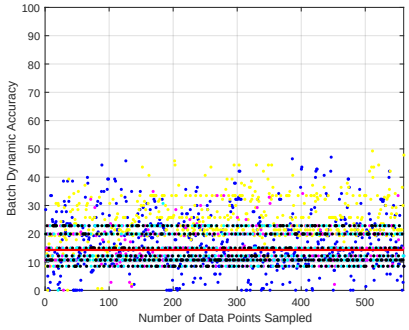


(g) CALM-nepLRB ($ITR = 1$)

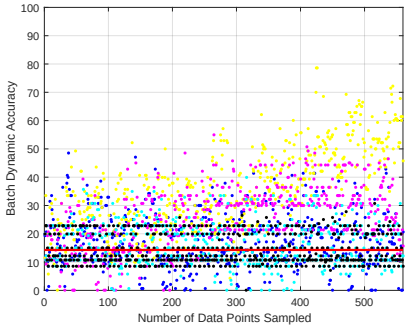


(h) CALM-nepLRB ($ITR = 200$)

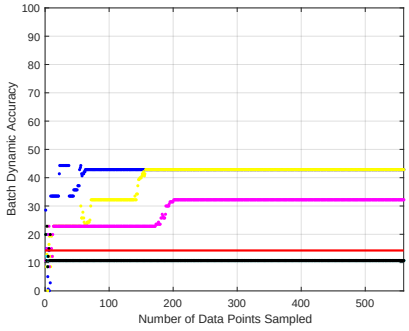
Figure 6.6: Accuracy on Data 3 - Training Fold 1



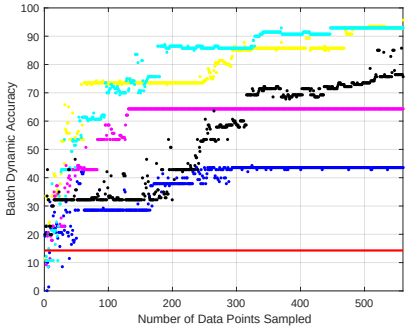
(a) CALM-rLRB ($ITR = 1$)



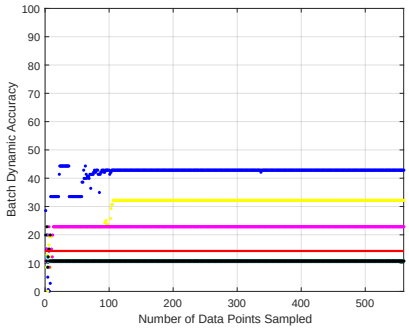
(b) CALM-rLRB ($ITR = 200$)



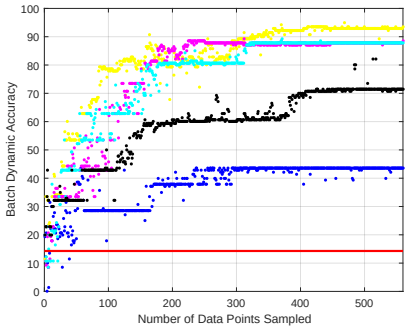
(c) CALM-eLRB ($ITR = 1$)



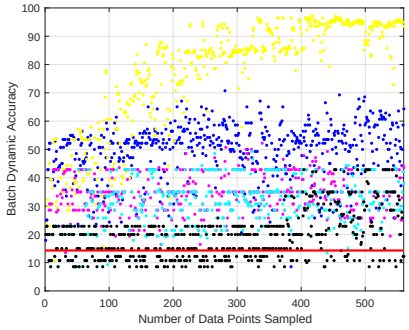
(d) CALM-eLRB ($ITR = 200$)



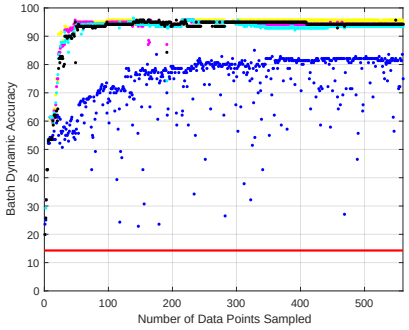
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

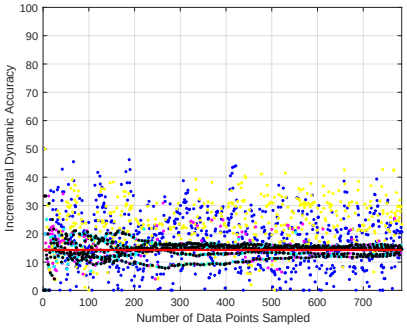


(g) CALM-nepLRB ($ITR = 1$)

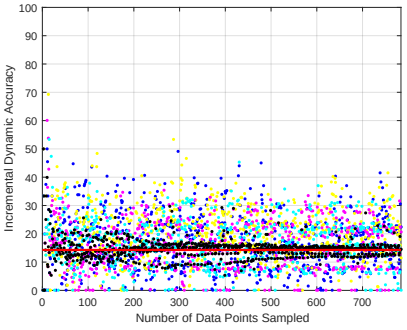


(h) CALM-nepLRB ($ITR = 200$)

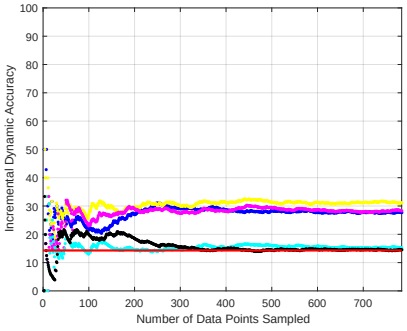
Figure 6.7: Accuracy on Data 3 - Testing Fold 1



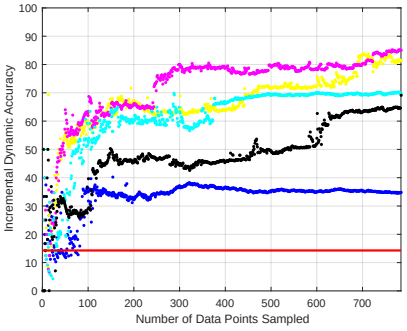
(a) CALM-rLRB ($ITR = 1$)



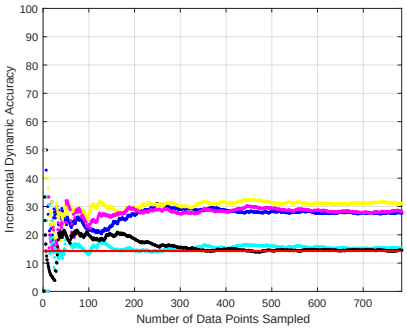
(b) CALM-rLRB ($ITR = 200$)



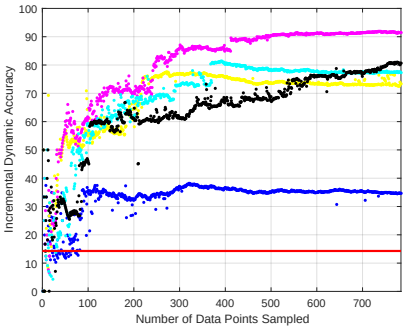
(c) CALM-eLRB ($ITR = 1$)



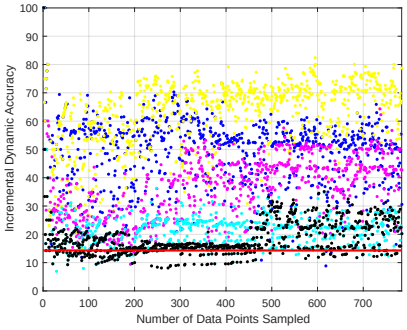
(d) CALM-eLRB ($ITR = 200$)



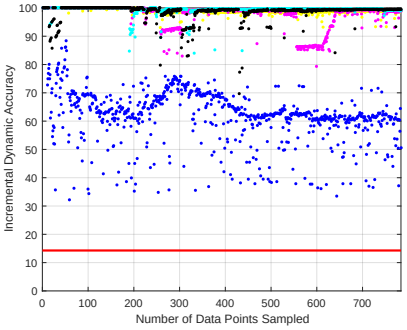
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

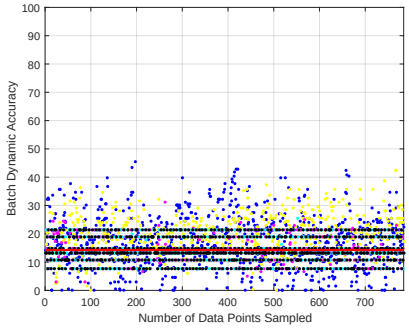


(g) CALM-nepLRB ($ITR = 1$)

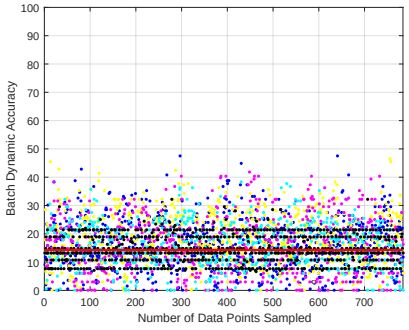


(h) CALM-nepLRB ($ITR = 200$)

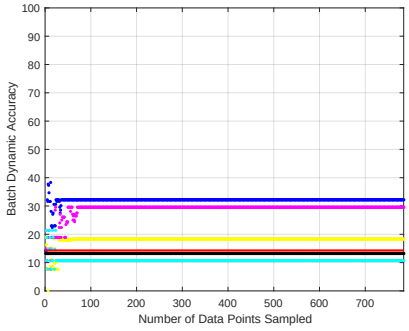
Figure 6.8: Accuracy on Data 4 - Training Fold 1



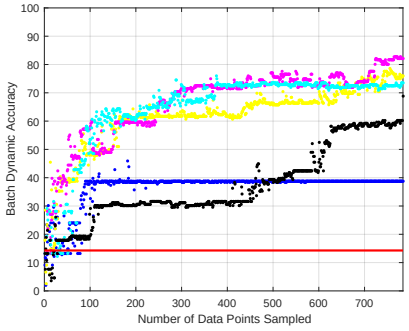
(a) CALM-rLRB ($ITR = 1$)



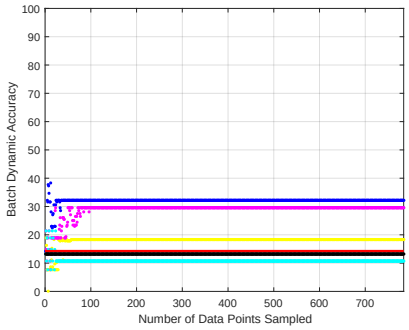
(b) CALM-rLRB ($ITR = 200$)



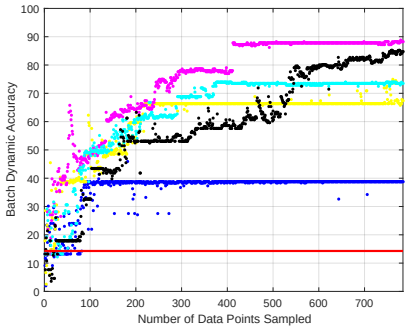
(c) CALM-eLRB ($ITR = 1$)



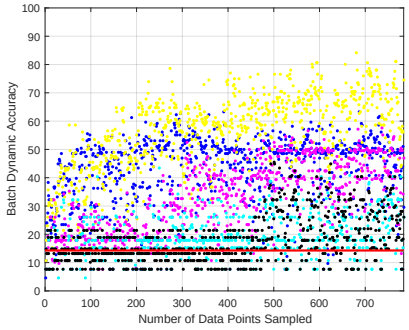
(d) CALM-eLRB ($ITR = 200$)



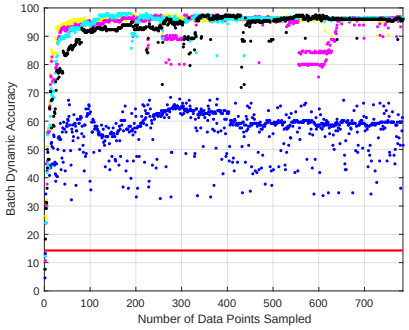
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

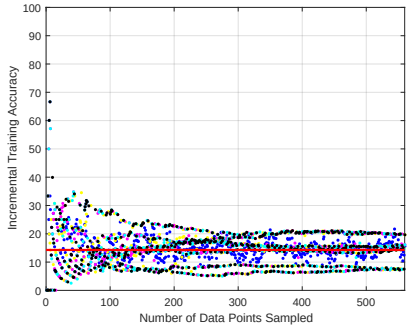


(g) CALM-nepLRB ($ITR = 1$)

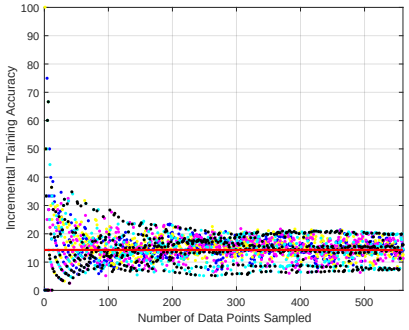


(h) CALM-nepLRB ($ITR = 200$)

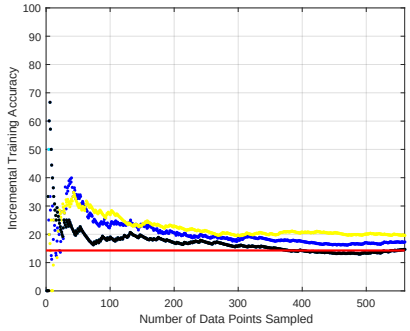
Figure 6.9: Accuracy on Data 4 - Testing Fold 1



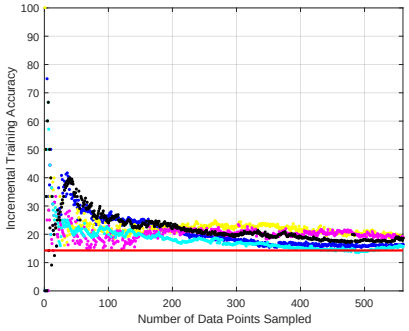
(a) CALM-rLRB ($ITR = 1$)



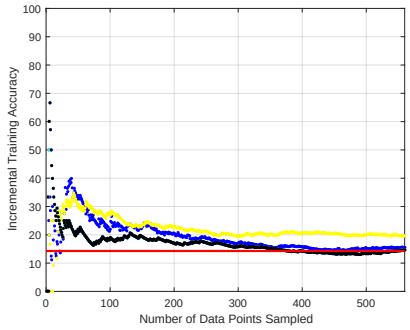
(b) CALM-rLRB ($ITR = 200$)



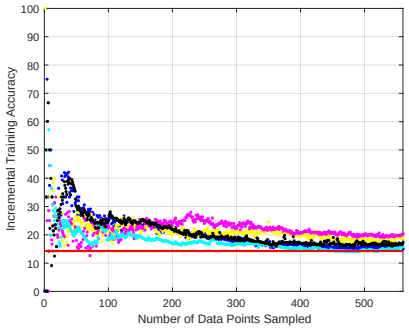
(c) CALM-eLRB ($ITR = 1$)



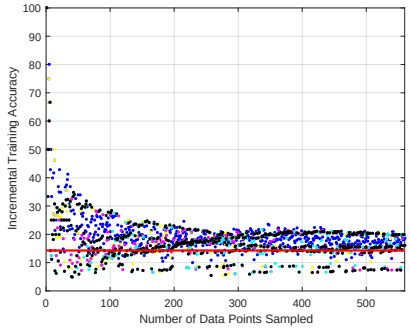
(d) CALM-eLRB ($ITR = 200$)



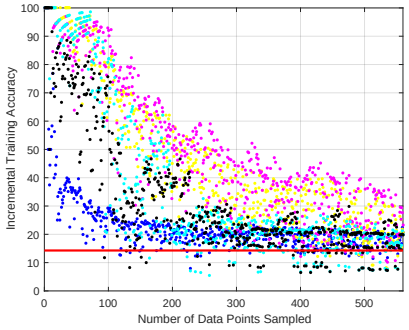
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

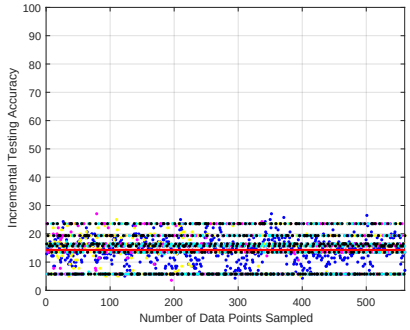


(g) CALM-nepLRB ($ITR = 1$)

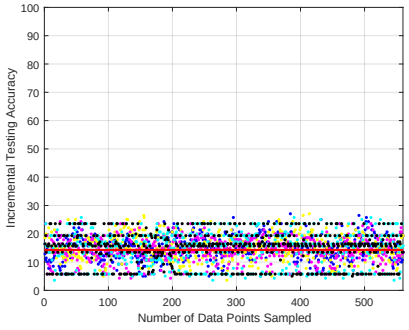


(h) CALM-nepLRB ($ITR = 200$)

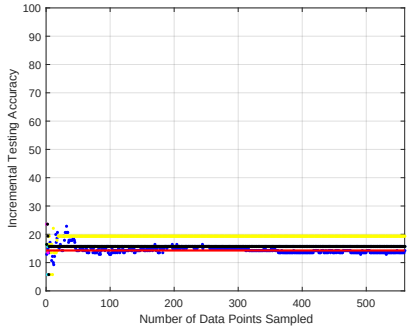
Figure 6.10: Accuracy on Data 5 - Training Fold 1



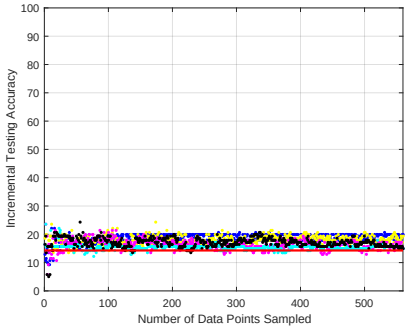
(a) CALM-rLRB ($ITR = 1$)



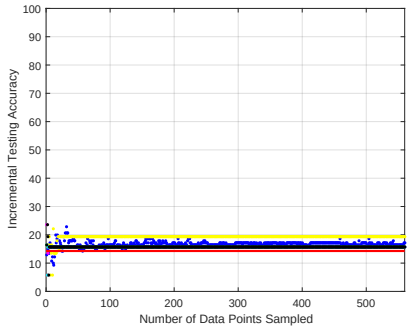
(b) CALM-rLRB ($ITR = 200$)



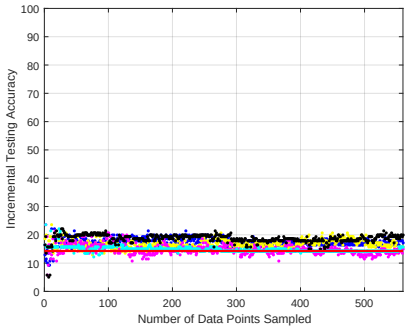
(c) CALM-eLRB ($ITR = 1$)



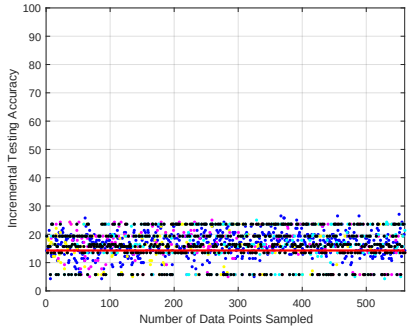
(d) CALM-eLRB ($ITR = 200$)



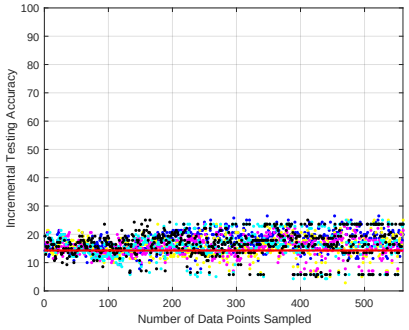
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)



(g) CALM-nepLRB ($ITR = 1$)



(h) CALM-nepLRB ($ITR = 200$)

Figure 6.11: Accuracy on Data 5 - Testing Fold 1

We will first look through the accuracy results from DATA1 to DATA4 and will see DATA5 in the last paragraph of this section since DATA5 is completely randomly distributed data.

According to the accuracies from Figure 6.2 to Figure 6.9, which corresponds from DATA1 to DATA4 respectively, we can see the accuracy of experience-based learning tends to gradually increase over learning steps when the number of iteration is 200 compared to the results when the iteration is 1. For example, the Figure 6.2d, Figure 6.2f, and Figure 6.2h shows more incremental increases on graphs than Figure 6.2c, Figure 6.2e, and Figure 6.2g. Moreover, those phenomenon similarly occurs on both training and testing data over all of the four data sets. This supports that using past positive experiences shows effective learning results with a certain number of iterations.

On the other hand, according to the figures from Figure 6.2 to Figure 6.9, we can see the number of iterations is not significantly important learning factor for CALM-rLRB since there are no striking accuracy growth. Moreover, there is no steady gradual growth in CALM-rLRB accuracy over learning steps. This means the optimization process with applying only current context is not enough to understand a given world and thus high accuracy is hard to be expected. However, it is notable that that CALM-rLRB with 3-layered CALM-ANN and 4-layered CALM-ANN shows improved accuracy with 200 iterations on DATA2 and DATA3. Especially, compared to the other CALM-ANNs, 3-layered CALM-ANN is exceptionally highly affected by the increased number of iteration on the data sets. Also, the accuracy is increased over learning steps as shown Figure 6.4b and Figure 6.6b. This is seen more clearly through the numerical values in Table 6.3 and Table 6.4. On Table 6.3, the accuracy of the CALM-rLRB with 3-layered CALM-ANN at the end of the learning step shows highest values

compared to the other different number of layered CALM-ANNs. This refers that DATA2, which has no overlapped clusters and exactly one cluster for 7 kinds of output, is appropriate input space for CALM-rLRB with 3-layered CALM-ANN to represent up to 76.8% accuracy. Similarly, in Table 6.4, we can see CALM-rLRB with 3-layered CALM-ANN also shows highest accuracy compared to the others. The accuracy for DATA3, which has 3 more clusters and overlapped clusters than DATA2, ranges from 37.1% to 69.3% while DATA2 varies from 54.1% to 76.8%. In this regard, we can infer that if the input data is clearly clustered and the distribution of clusters is not complex, CALM-rLRB, which only learn current context at each learning step, can understand a given world with a certain number of iterations. In other words, the performance of CALM-rLRB is highly depending on the given input problem. Except for this, the overall accuracy distributions of CALM-rLRB between one iterations and 200 iterations for all data sets are not strikingly different and thus it can be said that CALM-rLRB is less affected by the optimization regardless of the number of iterations since it only applies current context and reward information to the optimization process at each learning step.

Along with the same figures which used in second analytic conclusion, it is evident that using past positive experiences with a certain number of iterations overcomes the simple reward-based learning algorithm CALM-rLRB. More specifically, experience-based learning (CALM-eLRB, CALM-epLRB, and CALM-nepLRB) surpasses CALM-rLRB with a certain number of iterations. According to Table 6.2 through Table 6.5, all of the accuracies of experience-based learning are higher than the accuracies of CALM-rLRB in each different number of layered CALM-ANN in each fold.

Especially, above all, we can CALM-nepLRB outperforms both CALM-eLRB and CALM-epLRB with no regard to the number of iterations and depth of CALM-ANN. On figures from Figure 6.2 to Figure 6.9, when the number of iteration is 200, the increasing slope of the CALM-nepLRB accuracies is sharp than CALM-eLRB and CALM-epLRB. Also, when the number of iteration is 1, CALM-nepLRB shows relatively high accuracy while the others struggles for understanding a given world. On DATA2 and DATA3, 3-layered CALM-ANN of CALM-nepLRB shows over 90% accuracy while 3-layered CALM-ANN of others stays between 35% and 45%. This is seen more clearly in numerical values the tables from Table 6.2 to Table 6.5. Especially, in Table 6.3, CALM-nepLRB mostly shows 100% accuracy across the all number of layers and folds while CALM-epLRB and CALM-eLRB shows its 100% accuracy only with 3-layered and 5-layered CALM ANN; and CALM-rLRB shows its best performance, ranged from 54.1% to 76.8%, with its 3-layered CALM-ANN on DATA2. This phenomenon is similarly found on the other tables: Table 6.2, Table 6.4, and Table 6.5. In this regard, it is obvious that CALM-nepLRB is most accurate and CALM-rLRB is most basic algorithm while CALM-eLRB and CALM-epLRB shows comparable outcomes. Thus the algorithm effectiveness on accuracy analysis can be organized: CALM-rLRB $<$ CALM-eLRB, CALM-epLRB $<$ CALM-nepLRB.

In order to support the power of CALM-nepLRB, the reasons why CALM-nepLRB is most advanced algorithm can be organized with 3 experimental evidents; note that the following three facts are still based on DATA1, DATA2, DATA3, and DATA4. First, CALM-nepLRB shows more steep incremental accuracy compared to CALM-eLRB and CALM-epLRB. This means that it is aware of contexts faster than the others given same conditional variables. For

example, as shown figures from Figure 6.2 to Figure 6.9, CALM-nepLRB reaches 80% accuracy before learning step 50 across all of data sets and all kinds number of layers except for 2-layered CALM-nepLRB-ANN. On the other hand, CALM-eLRB and CALM-epLRB can not their performance over 80% accuracy within 50 number of learning steps. Moreover, the final highest accuracy at the end of the learning step of CALM-nepLRB for all data sets is always greater than or equals to 95% based in Table 6.2, Table 6.3, Table 6.4, Table 6.5, and Table 6.6. For example, on DATA2 as shown Table 6.3, CALM-nepLRB shows 100% accuracy for all folds except for 2-layered CALM-nepLRB-ANN, which means it can completely recognize given input space, DATA2, with networks over two layers. Second, CALM-nepLRB can make relatively higher performance even with only one iteration compared to the others. For example, on DATA2, 2-layered and 3-layered CALM-nepLRB-ANN shows over 90% accuracy with only one iteration. Also, on DATA3, 3-layered CALM-nepLRB-ANN also shows over 90% accuracy when the number of iteration is 1. On both DATA1 and DATA4, the accuracy does not reach 90% but still shows higher figures compared to the other threes. Therefore, CALM-nepLRB needs less number of iterations which can reduce the computational complexity of the learning process. Third, in CALM-nepLRB algorithm, even the simplest network, 2-layered CALM-nepLRB-ANN, show less limitations on representing the input space compared to the other algorithms. It is very notable that the 2-layered neural network shows the limitations on classifying all the data sets, meaning having only input and output layer is not enough to represent all the given input space. This can be supported by the numerical values through the same tables; we can see the accuracy of 2-layered CALM-nepLRB-ANN outperforms the 2-layered CALM-ANNs of all the other three algorithms. From these reasons, CALM-nepLRB is outperforming algorithm relative to the other algorithms; also it generally gives high-performance

which is supported by the figures and numerical values on the tables.

Note that the accuracy results form both training data and testing data are similar across all the data sets and all different number of layered neural networks. This supports the reliability of the learning process of CALM.

DATA5 is special data set as it is highly random distributed. In other words, it represents extremely unstructured and non-predictive contexts in the world. As a results, Figure 6.11 shows that none of CALM algorithms can give plausible accuracy on testing data set of DATA5 through the all learning steps. However, on training process as shown Figure 6.10, we can see in the early learning steps the algorithms shows better performance than the end of learning steps; especially, CALM-nepLRB shows very high performance in the beginning of the learning process but gets confused over the learning steps like the others. But note that CALM-nepLRB shows slow decreasing of the accuracy than the others; this means CALM-nepLRB learns faster and get confused slower than the others.

From the experimental results of the accuracy measurement from both training and testing data, five overall analytic conclusions can be summarized as follows. First, in terms of experience-based learning (CALM-eLRB, CALM-epLRB, CALM-nepLRB), them number of iterations are an important factor for highly accurate context-awareness since the experience-based learning can more optimize its CALM-ANN with the given number of iterations at each learning step. Second, CALM-rLRB is barely affected by the change of the number of iteration since it only focuses on current context and current reward value without any past experiences. Third, the experience-based learning (CALM-

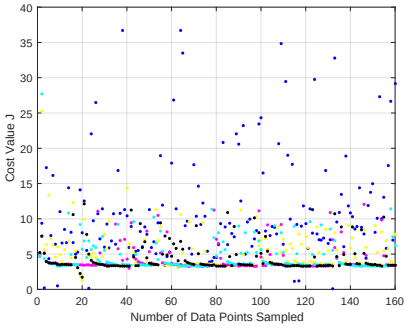
eLRB, CALM-epLRB, CALM-nepLRB) overcomes the reward-based learning (CALM-rLRB) as more past positive experiences are memorized over time with a certain number of iteration. Fourth, CALM-nepLRB outperforms the other algorithms with highest accuracy and it shows relatively high accuracy even when the number of iteration is 1, which supports the benefits of additional natural animal neurobiological features. Fifth, it can be considered that the accuracy analysis is reliable since there is little differences between training and testing accuracies as shown in Table 6.2, Table 6.3, Table 6.4, Table 6.5, and Table 6.6.

6.2.2 Cost Function Values Analysis

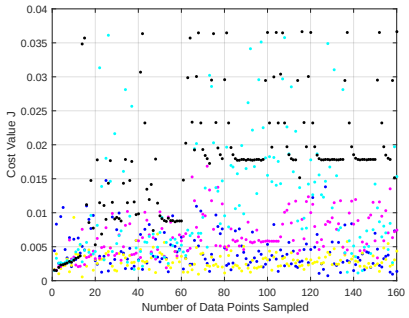
The second measurement of CALM performance is checking cost values over learning steps. By looking into cost function values at each learning step, we can see how each learning algorithm optimizes its neural networks effectively. As we discussed the role of cost function of supervised neural learning model in Section 3.2, if the learning process is to be successful, the cost function values should be decreased over learning steps since the errors between actual output and quasi-target output are expected to be decreased as a learning effect. In this section, we will see how each learning algorithm of CALM learns a given world by analyzing its cost function values over learning steps in each data set. Note that this cost values are calculated during the learning process thus the results of costs values are based on training data, not testing data. Likewise, the training data includes the 80% of the original data set so the number of input data is different from the original data points.

Figure 6.12, Figure 6.13, Figure 6.14, Figure 6.15, and Figure 6.16 show the cost function values of CALM algorithms with five different depths of CALM-ANN on fold1 of DATA1, DATA2, DATA3, DATA4, and DATA5 respectively. Each data point refers to the cost function value of a learning algorithm over the incremental input contexts from training data at each learning step. Specifically, each data point refers to the value of $J(t)$ which is the result of CALM-LRB-CORE or CALM-LRB-CORE-GEN in Algorithm 10, Algorithm 12, Algorithm 14, and Algorithm 16. Note that the range of y-axis for each graph is set differently; this is because the maximum cost value for each algorithm on each data is not consistency and thus having different y-axis range helps to probe each graph. For example, In Figure 6.12a, y-axis for CALM-rLRB without iteration is from 0 to 40 while y-axis for both CALM-rLRB with 200 number of iterations ranges from 0 to 0.04 as shown in Figure 6.12b. In this case, if the y-axis ranges are set from 0 to 40 for both of them, the graph for CALM-rLRB with 200 is hard to visually analyze its changes of cost values over learning steps since its actual ranges are all below 0.05.

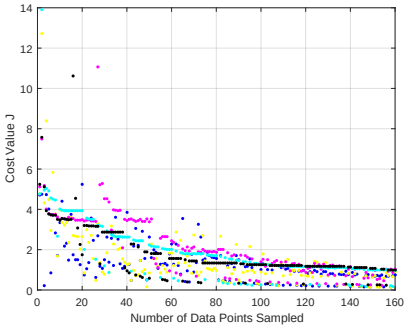
Note that with the 200 number of iterations CALM-rLRB shows very tiny cost function value (under 0.05) through all data sets: DATA1, DATA2, DATA3, DATA4, and DATA5; this is because CALM-rLRB optimizes only current context in CALM-LRB-CORE and thus, in each learning step, it can highly reduce its errors between the current context and corresponding quasi-target output compared to the other algorithms which optimize their accumulated experiences. Similarly, with the same reason, CALM-rLRB can not gradually decrease its cost value over the learning steps as it utilizes only current context in its optimization process.



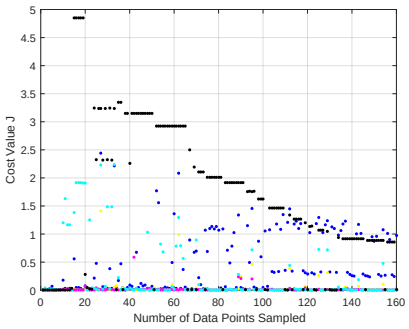
(a) CALM-rLRB ($ITR = 1$)



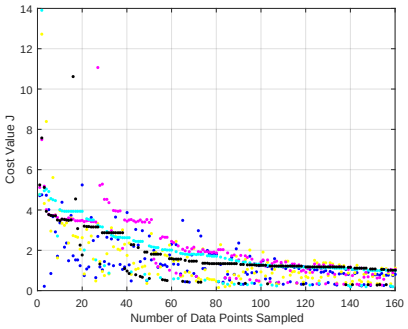
(b) CALM-rLRB ($ITR = 200$)



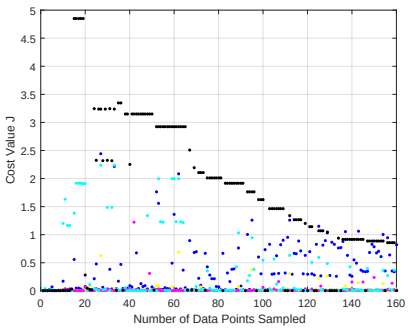
(c) CALM-eLRB ($ITR = 1$)



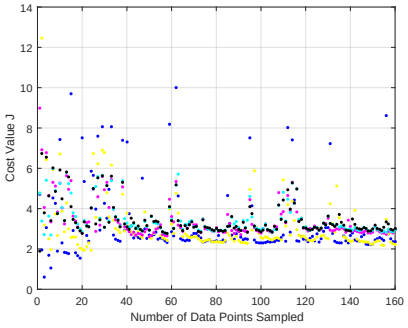
(d) CALM-eLRB ($ITR = 200$)



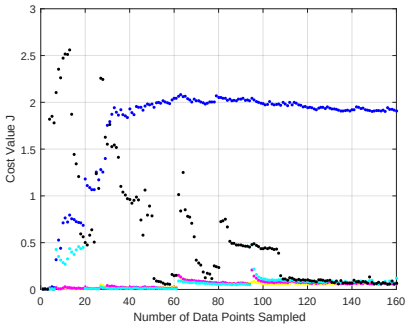
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

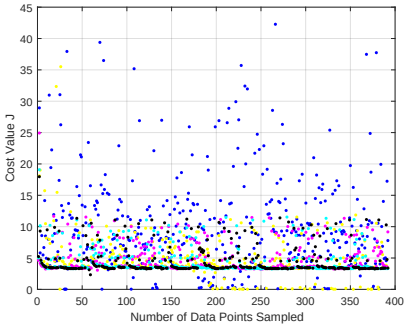


(g) CALM-nepLRB ($ITR = 1$)

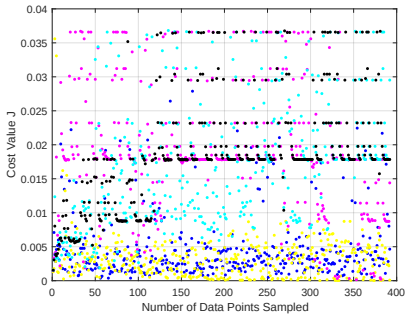


(h) CALM-nepLRB ($ITR = 200$)

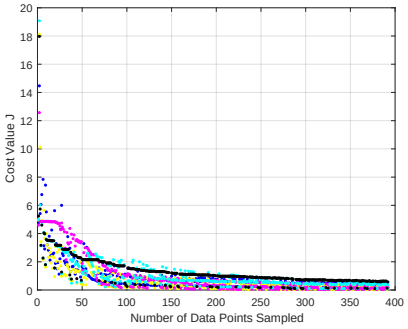
Figure 6.12: Cost Function Values on Data 1 (Fold 1)



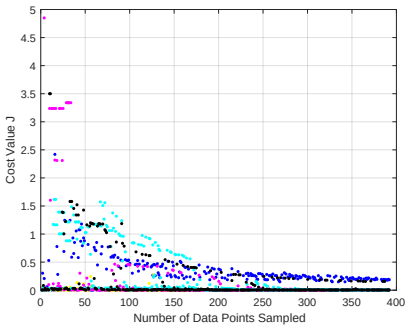
(a) CALM-rLRB ($ITR = 1$)



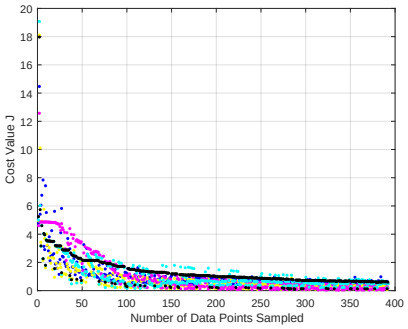
(b) CALM-rLRB ($ITR = 200$)



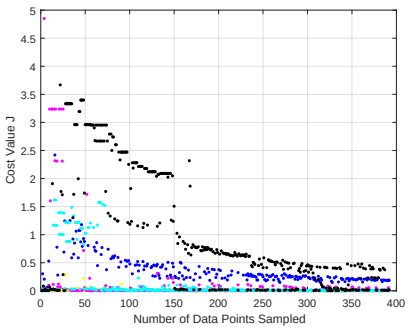
(c) CALM-eLRB ($ITR = 1$)



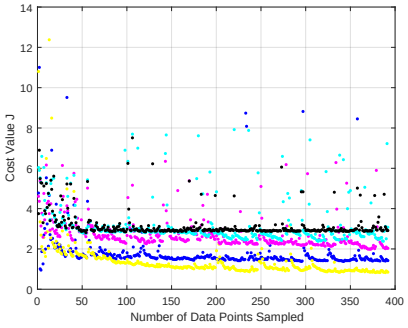
(d) CALM-eLRB ($ITR = 200$)



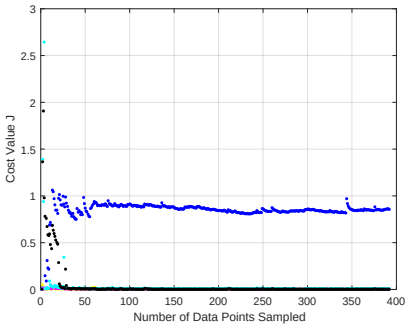
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

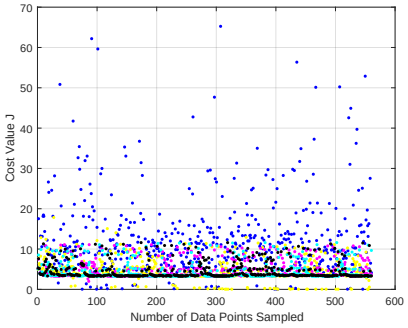


(g) CALM-nepLRB ($ITR = 1$)

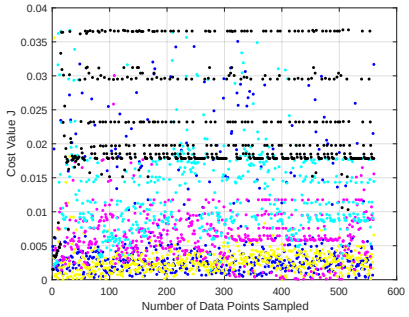


(h) CALM-nepLRB ($ITR = 200$)

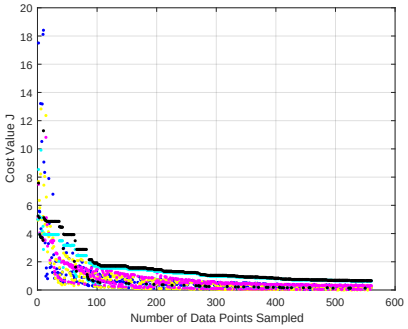
Figure 6.13: Cost Function Values on Data 2 (Fold 1)



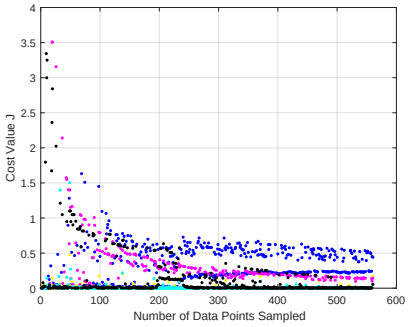
(a) CALM-rLRB ($ITR = 1$)



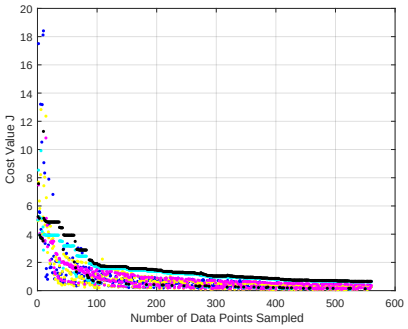
(b) CALM-rLRB ($ITR = 200$)



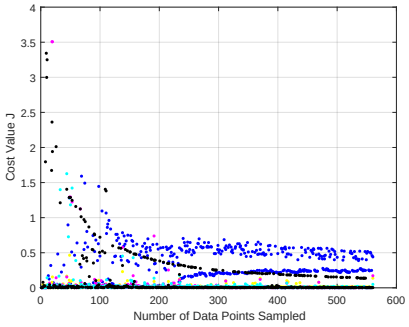
(c) CALM-eLRB ($ITR = 1$)



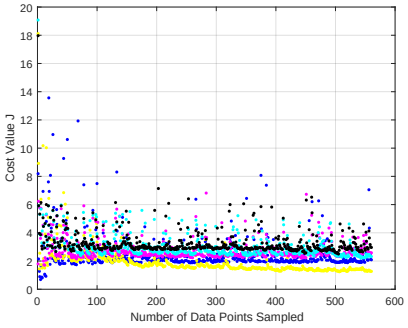
(d) CALM-eLRB ($ITR = 200$)



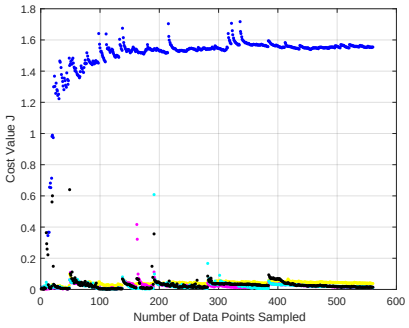
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

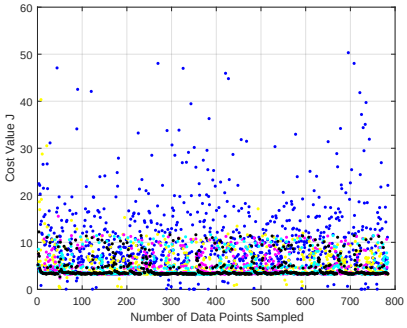


(g) CALM-nepLRB ($ITR = 1$)

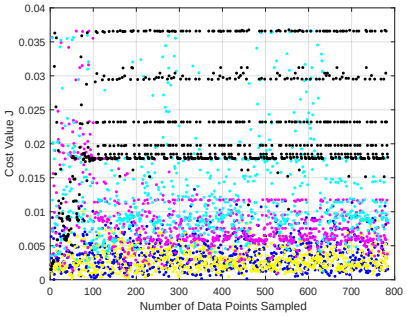


(h) CALM-nepLRB ($ITR = 200$)

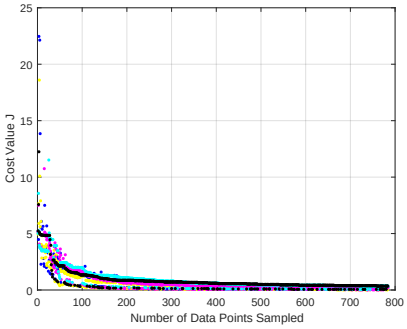
Figure 6.14: Cost Function Values on Data 3 (Fold 1)



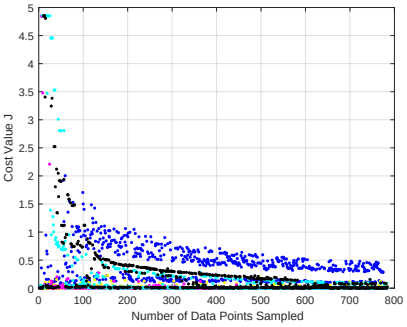
(a) CALM-rLRB ($ITR = 1$)



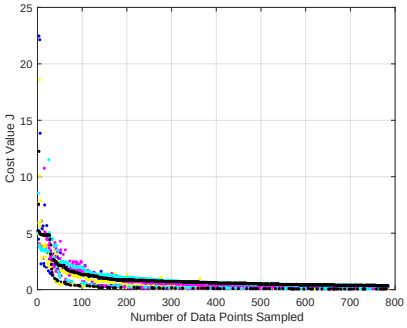
(b) CALM-rLRB ($ITR = 200$)



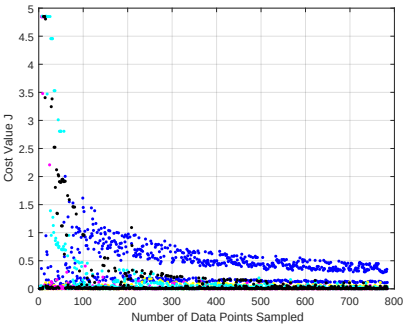
(c) CALM-eLRB ($ITR = 1$)



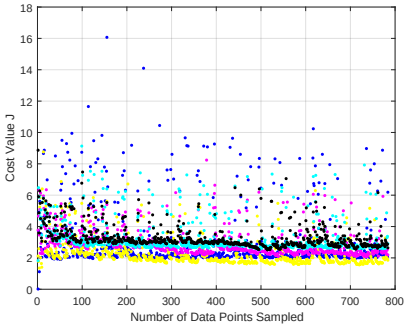
(d) CALM-eLRB ($ITR = 200$)



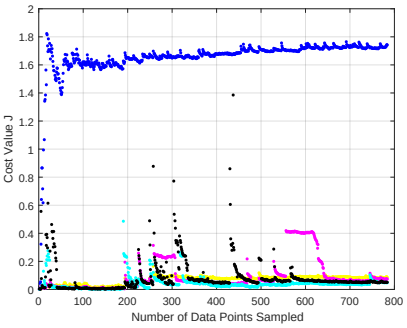
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

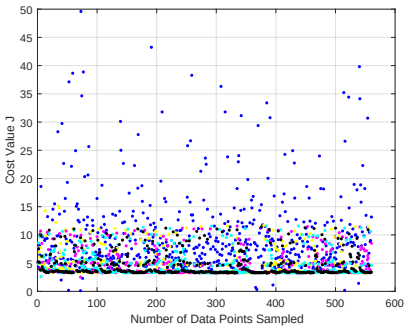


(g) CALM-nepLRB ($ITR = 1$)

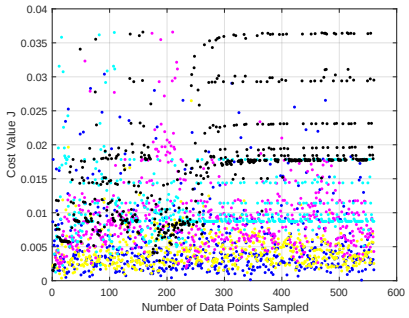


(h) CALM-nepLRB ($ITR = 200$)

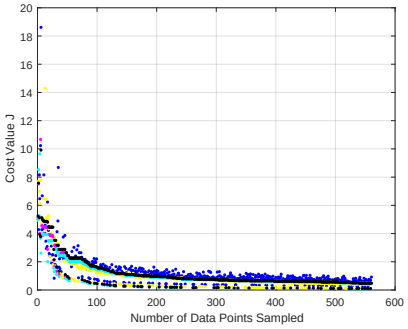
Figure 6.15: Cost Function Values on Data 4 (Fold 1)



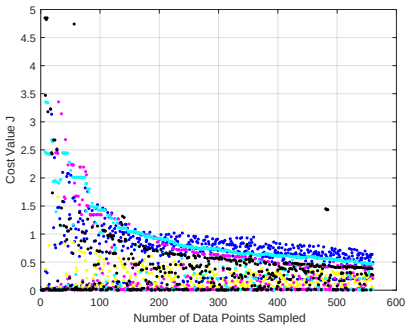
(a) CALM-rLRB ($ITR = 1$)



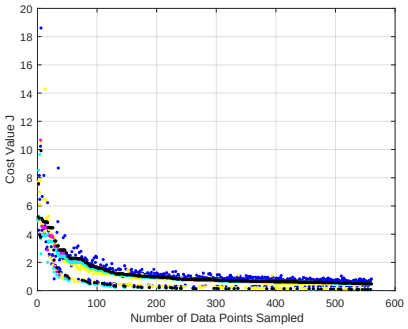
(b) CALM-rLRB ($ITR = 200$)



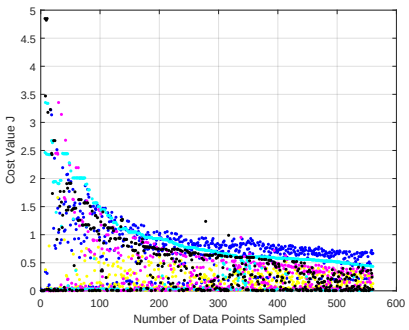
(c) CALM-eLRB ($ITR = 1$)



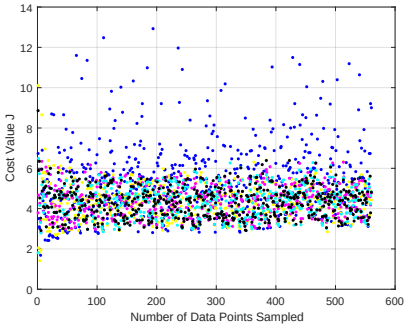
(d) CALM-eLRB ($ITR = 200$)



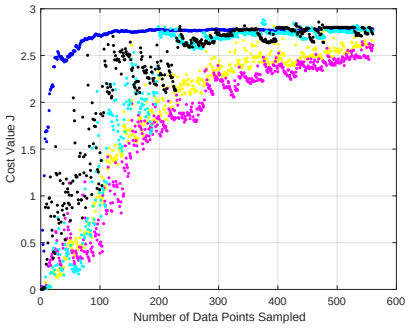
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)



(g) CALM-nepLRB ($ITR = 1$)



(h) CALM-nepLRB ($ITR = 200$)

Figure 6.16: Cost Function Values on Data 5 (Fold 1)

The first salient feature on this measurement is that each CALM algorithm with each different CALM-ANN in each data set evidently reduces its range of cost values when it has 200 number of iteration. For example, in Figure 6.12, the cost value range of CALM-eLRB and CALM-epLRB is from 0 to 14 with only one iteration while it is from from 0 to 5 with 200 number of iteration. This phenomenon is in common with the other cases over the figures from Figure 6.12 to Figure 6.16. This implies that CALM better optimize its neural networks over learning process with a certain number of iteration by reducing errors between actual and quasi-target output. This supports the result from accuracy analysis such as having a certain number of iterations shows higher accuracy than having only one iteration.

CALM-eLRB and CALM-epLRB algorithms mostly show monotonic decrease of the cost function values over the learning steps. This means they reduce errors between input context and quasi-target over learning steps as a result of learning process. However, for CALM-rLRB, cost values of each different CALM-ANN is not one directional and distributed highly unevenly. This means the errors are not gradually diminished over the learning steps and this supports the reason why the accuracy of CALM-rLRB on each data set are not steadily increased. CALM-nepLRB shows interesting form of cost function values where its accuracy is almost around 99% except for on DATA5 as we covered in Chapter 6.2.1. First of all, when the number of iteration is 1, interestingly, for all data sets we can see the cost function graphs of CALM-rLRB and CALM-nepLRB shows similar patterns in some degree. This is because CALM-nepLRB has flexibility to switch algorithm between CALM-rLRB and CALM-epLRB based on recurrent inhibition and neuromodulatory process, especially dopaminergic process. Note that CALM-nepLRB takes back the currently performed behavior if the reward

is negative and exploits CALM-eLRB learning process when the number of non-rewarding behavior is over the recurrent threshold. We covered how this is carried out neurobiologically in a newly designed generalized, arbitrary-depth, neural network, CALM-nepLRB-ANN. In this regard, if CALM-nepLRB takes CALM-rLRB algorithm more often as the recurrent inhibition arises frequently, the cost functions for both algorithm can have similar patterns. On the other hand, when the number of iteration is 200, CALM-nepLRB shows much more reduced cost values over the learning steps compared when the only one iteration, which supports the accuracy with 200 iteration is higher than itself with the only one iteration. Moreover, except for 2-layered CALM-nepLRB-ANN, most of cost functions keeps very low value, almost closer to zero, on DATA2, DATA3, and DATA4. On the other hand, the 2-layered CALM-nepLRB-ANN shows higher cost value than the other layers through all of the data sets, which supports the accuracy of 2-layered CALM-nepLRB-ANN is generally lower then the other number of layered CALM-nepLRB-ANNs.

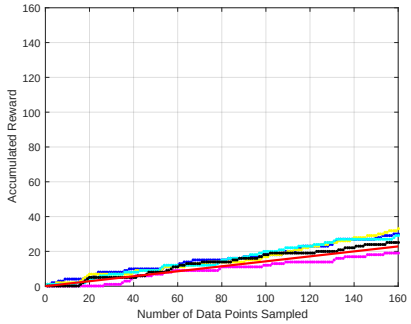
Also there are unique pattern of CALM-nepLRB with 200 iterations, which shows acute vertical increase at several points of over learning steps. For example, in Figure 6.15h, when the learning step is 200, we can see the sharp increase of cost value. This implies at this learning step, CALM-nepLRB selects CALM-rLRB learning algorithm thus the summation of total number of errors at the learning step is increased. In other words, when CALM-nepLRB exploits CALM-rLRB, the error between current input context and quasi-target output will be reduced but the total errors between all accumulated past contexts and quasi-target outputs will be increased. Note that the reason selecting CALM-rLRB is to avoid currently repeatedly selecting incorrect behavior, not to optimize current neural network for all input world.

In Figure 6.16h, we can see the cost function values for all layers are increasing instead of decreasing. This means the errors are increasing over learning steps since the DATA5 feeds highly unstructured incremental data points. This is supported by the decreasing accuracy of CALM-nepLRB over learning steps as shown in Figure 6.10h.

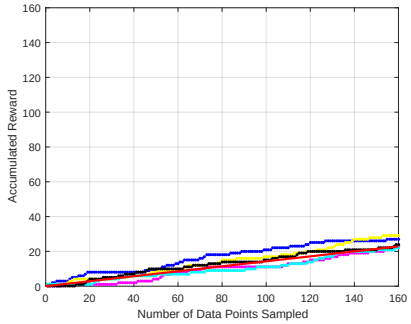
6.2.3 Accumulated Rewards Analysis

The third measurement of CALM performance is to check accumulated rewards over the learning steps. By reviewing the behavior of accumulated rewards, we can see how each CALM algorithm seeks the right output corresponding a given context at each learning step as well as the total number of rewards over the learning steps. Ideally, if a reward-based learning model is efficient, it shows increasing accumulated rewards over time which implies the model better understand a given world over time.

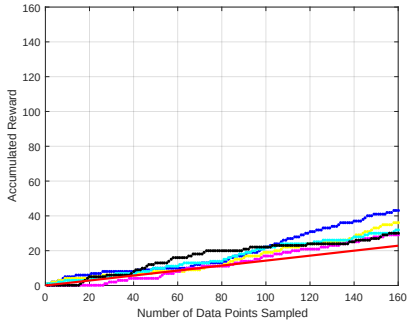
Figure 6.17, Figure 6.18, Figure 6.19, Figure 6.20, and Figure 6.21 show the accumulated rewards values of each CALM algorithm with five different depths of CALM-ANN on fold1 of DATA1, DATA2, DATA3, DATA4, and DATA5 respectively. Each data point refers to the number of rewards which has been accumulated from the initial learning step to the current incremental learning steps. Note that these results are also from training data since the experiments for accumulated rewards are to check how CALM utilizes reward value during the learning progress over time.



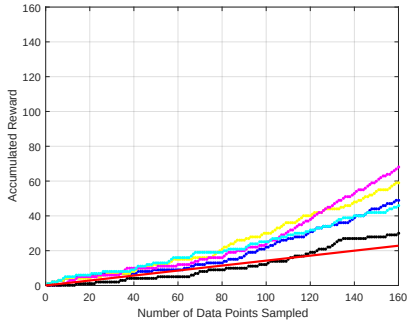
(a) CALM-rLRB ($ITR = 1$)



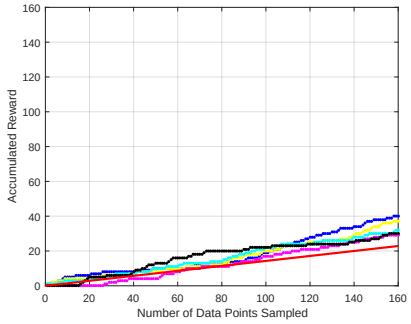
(b) CALM-rLRB ($ITR = 200$)



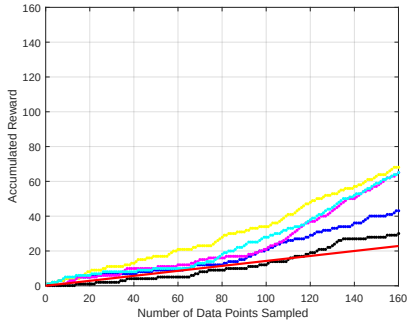
(c) CALM-eLRB ($ITR = 1$)



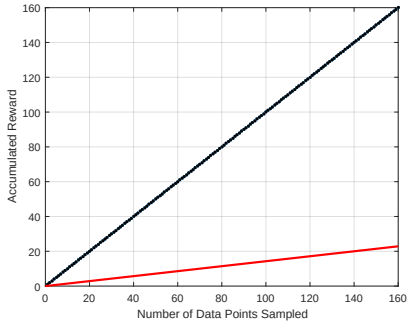
(d) CALM-eLRB ($ITR = 200$)



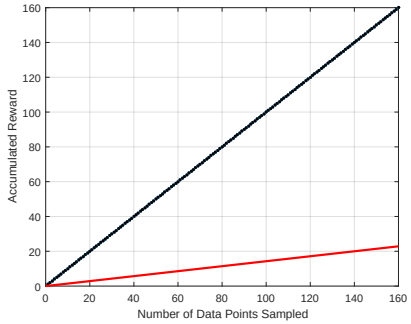
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

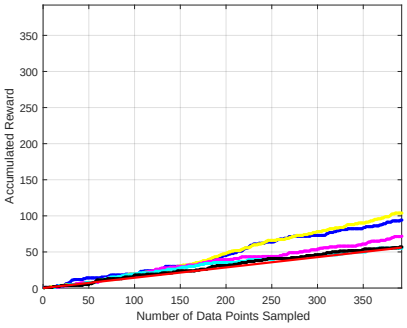


(g) CALM-nepLRB ($ITR = 1$)

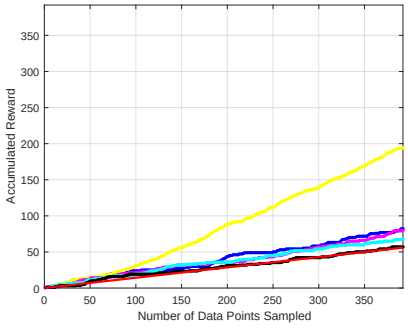


(h) CALM-nepLRB ($ITR = 200$)

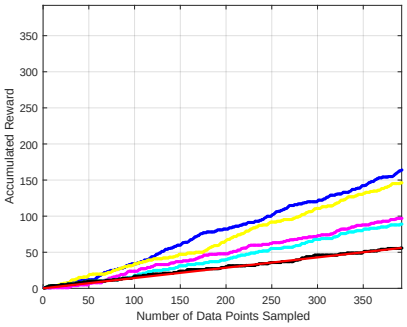
Figure 6.17: Accumulated Rewards on Data 1 (Fold 1)



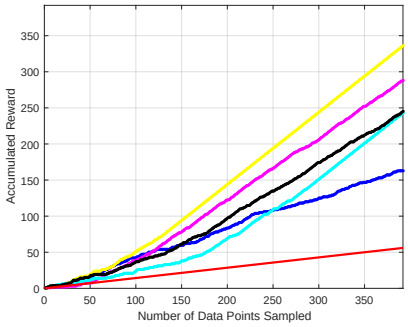
(a) CALM-rLRB ($ITR = 1$)



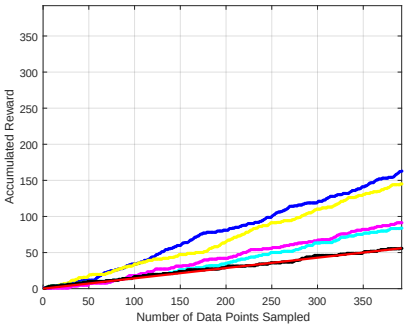
(b) CALM-rLRB ($ITR = 200$)



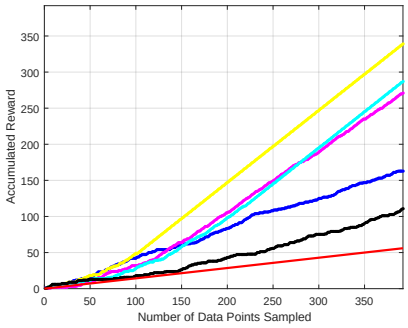
(c) CALM-eLRB ($ITR = 1$)



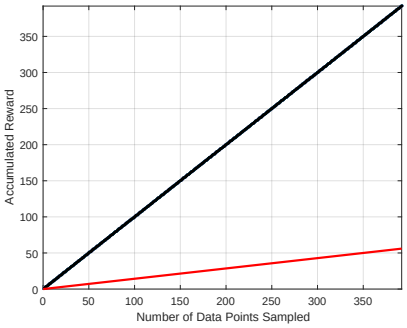
(d) CALM-eLRB ($ITR = 200$)



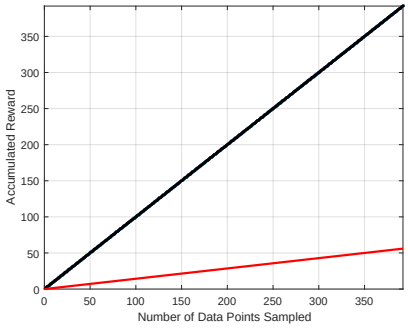
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

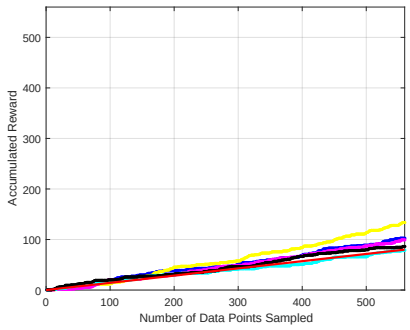


(g) CALM-nepLRB ($ITR = 1$)

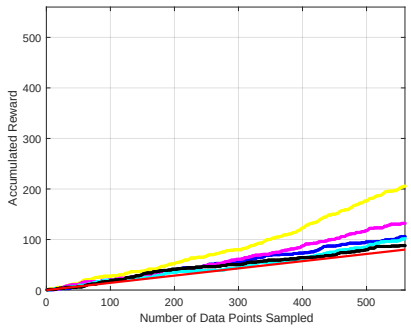


(h) CALM-nepLRB ($ITR = 200$)

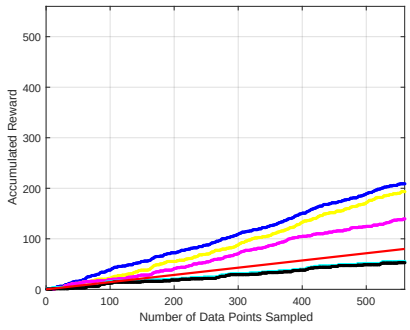
Figure 6.18: Accumulated Rewards on Data 2 (Fold 1)



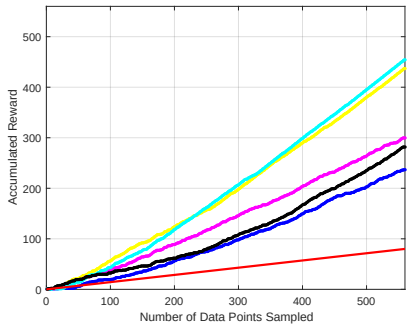
(a) CALM-rLRB ($ITR = 1$)



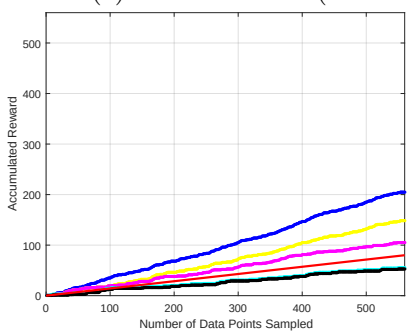
(b) CALM-rLRB ($ITR = 200$)



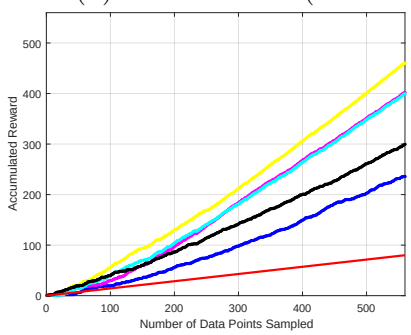
(c) CALM-eLRB ($ITR = 1$)



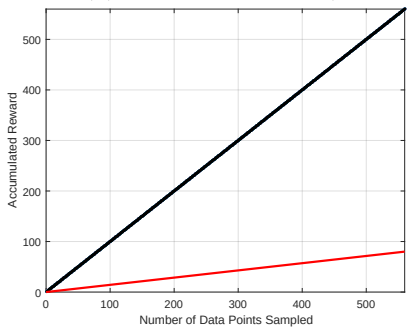
(d) CALM-eLRB ($ITR = 200$)



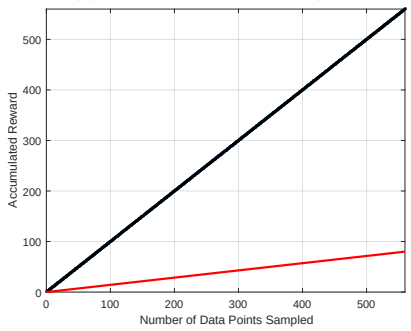
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

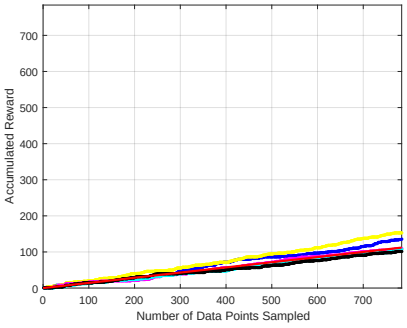


(g) CALM-nepLRB ($ITR = 1$)

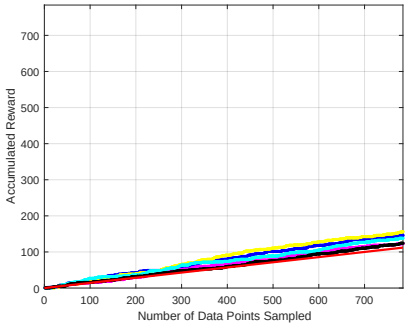


(h) CALM-nepLRB ($ITR = 200$)

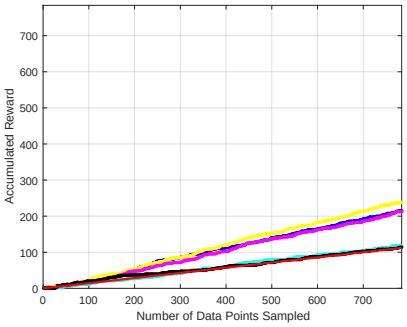
Figure 6.19: Accumulated Rewards on Data 3 (Fold 1)



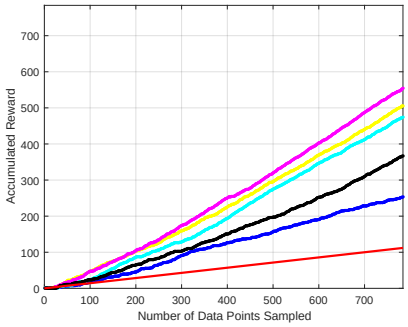
(a) CALM-rLRB ($ITR = 1$)



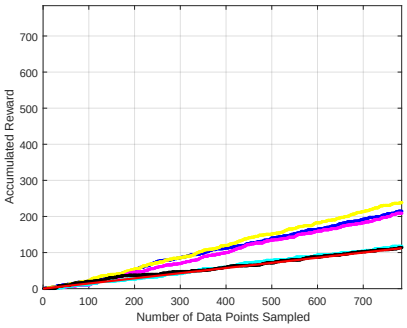
(b) CALM-rLRB ($ITR = 200$)



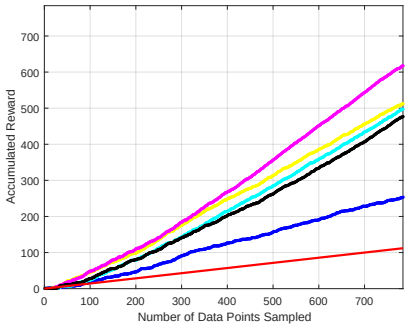
(c) CALM-eLRB ($ITR = 1$)



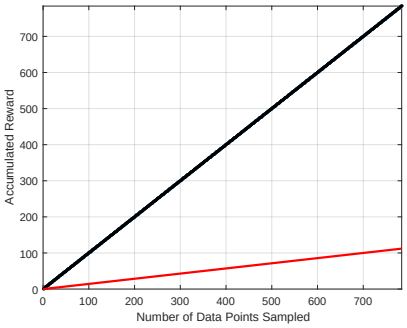
(d) CALM-eLRB ($ITR = 200$)



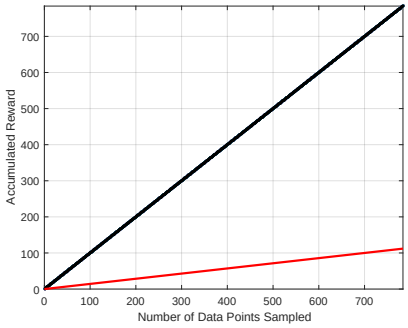
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

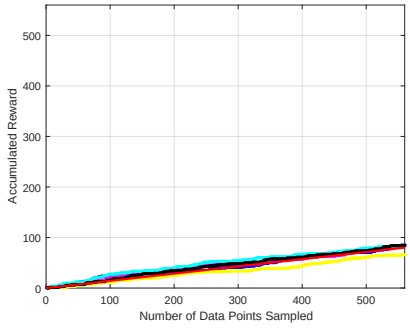


(g) CALM-nepLRB ($ITR = 1$)

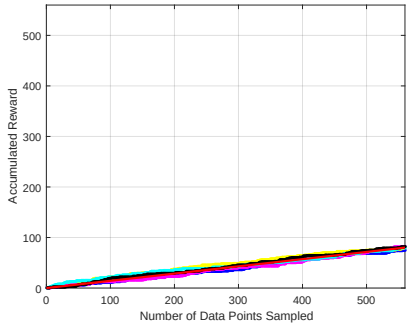


(h) CALM-nepLRB ($ITR = 200$)

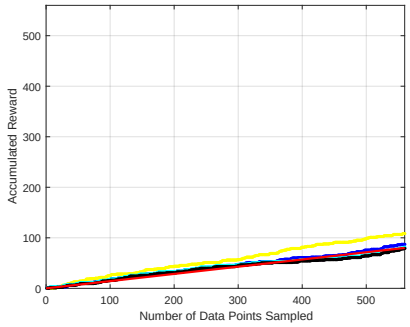
Figure 6.20: Accumulated Rewards on Data 4 (Fold 1)



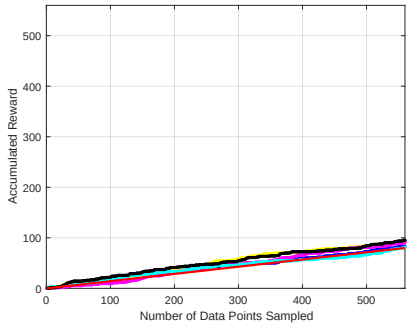
(a) CALM-rLRB ($ITR = 1$)



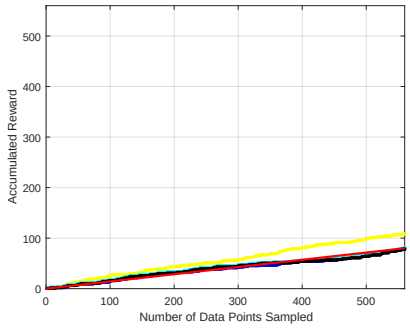
(b) CALM-rLRB ($ITR = 200$)



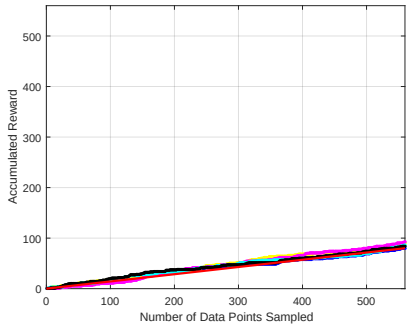
(c) CALM-eLRB ($ITR = 1$)



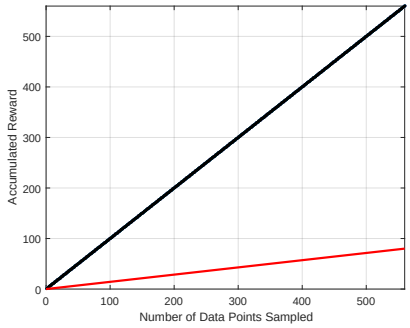
(d) CALM-eLRB ($ITR = 200$)



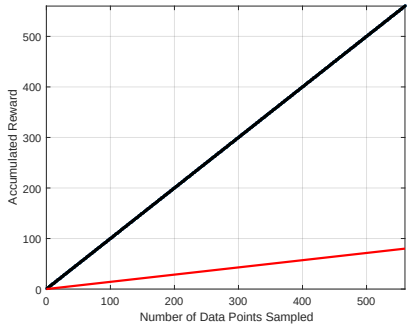
(e) CALM-epLRB ($ITR = 1$)



(f) CALM-epLRB ($ITR = 200$)

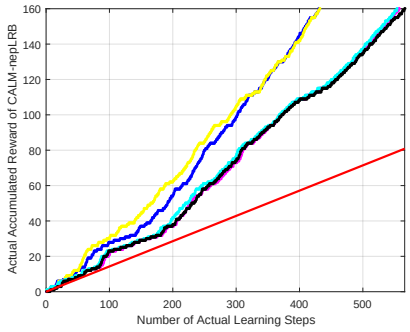


(g) CALM-nepLRB ($ITR = 1$)

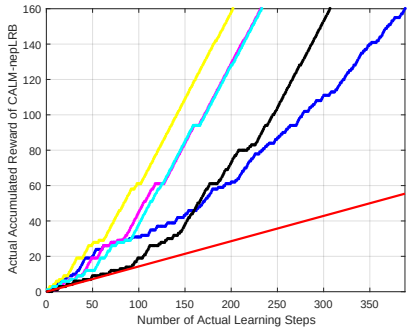


(h) CALM-nepLRB ($ITR = 200$)

Figure 6.21: Accumulated Rewards on Data 5 (Fold 1)

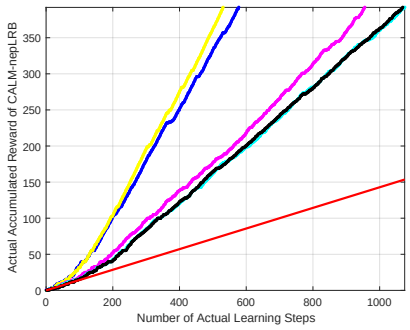


(a) CALM-nepLRB ($ITR = 1$)

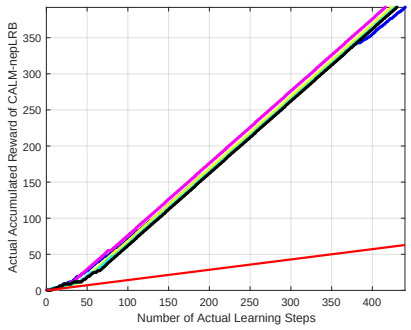


(b) CALM-nepLRB ($ITR = 200$)

Figure 6.22: Actual Accumulated Rewards in CALM-nepLRB on DATA1 (Fold 1)

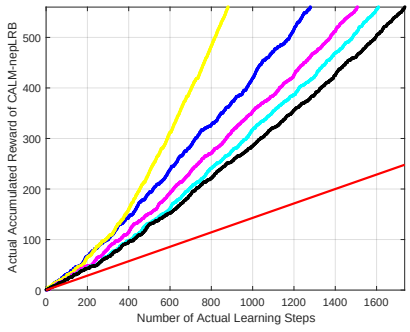


(a) CALM-nepLRB ($ITR = 1$)

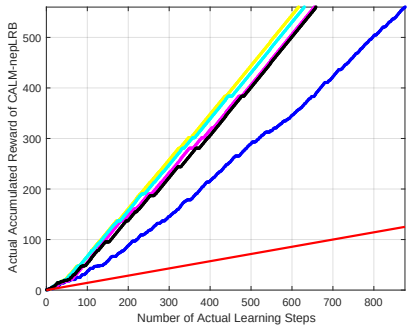


(b) CALM-nepLRB ($ITR = 200$)

Figure 6.23: Actual Accumulated Rewards in CALM-nepLRB on DATA2 (Fold 1)

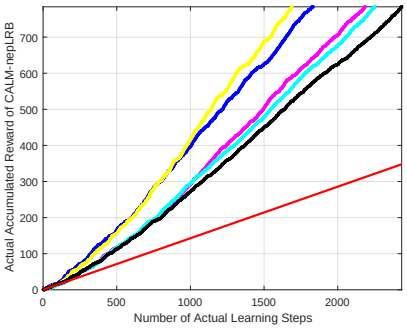


(a) CALM-nepLRB ($ITR = 1$)

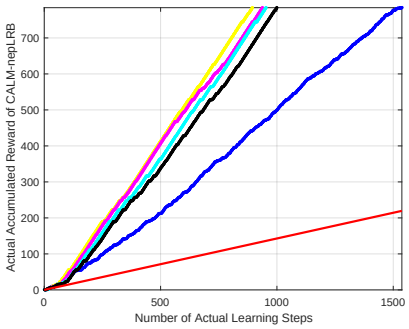


(b) CALM-nepLRB ($ITR = 200$)

Figure 6.24: Actual Accumulated Rewards in CALM-nepLRB on DATA3 (Fold 1)

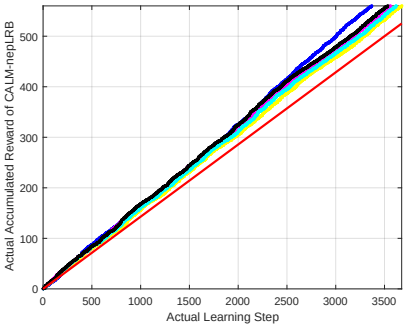


(a) CALM-nepLRB ($ITR = 1$)

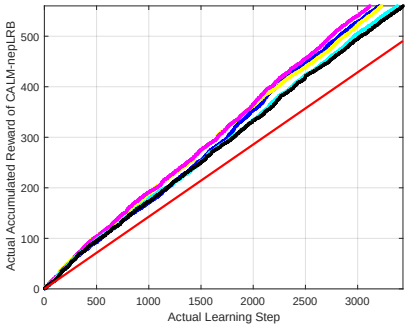


(b) CALM-nepLRB ($ITR = 200$)

Figure 6.25: Actual Accumulated Rewards in CALM-nepLRB on DATA4 (Fold 1)



(a) CALM-nepLRB ($ITR = 1$)



(b) CALM-nepLRB ($ITR = 200$)

Figure 6.26: Actual Accumulated Rewards in CALM-nepLRB on DATA5 (Fold 1)

It is notable that all accumulated rewards are increased over learning steps for all algorithms on all data sets with regardless of the number of iterations. This means each learning algorithm adjusts its neural network towards getting reward from an environment. More strikingly, CALM-nepLRB always gets reward at each learning step since it takes neuromodulation and recurrent inhibition. For example, at learning step t, if recurrent inhibition is required then CALM-nepLRB takes back the original behavior so that it can face to same context again and then performs CALM-rLRB, which guarantees to find right behavior for the current context eventually. This means, only in CALM-nepLRB, the total number of actual learning steps including the expanded learning steps of withdrawing behaviors is larger than the number of data points to be learned. In this regard, Figure 6.22, Figure 6.23, Figure 6.24, Figure 6.25, and Figure 6.26 shows the actual accumulated rewards over expanded learning steps, which is feasible only in CALM-nepLRB. In these figures, we can see CALM-nepLRB guarantee to get a reward at each learning step including the extended repeated learning steps; thus the actual learning steps are greater than the number of data points.

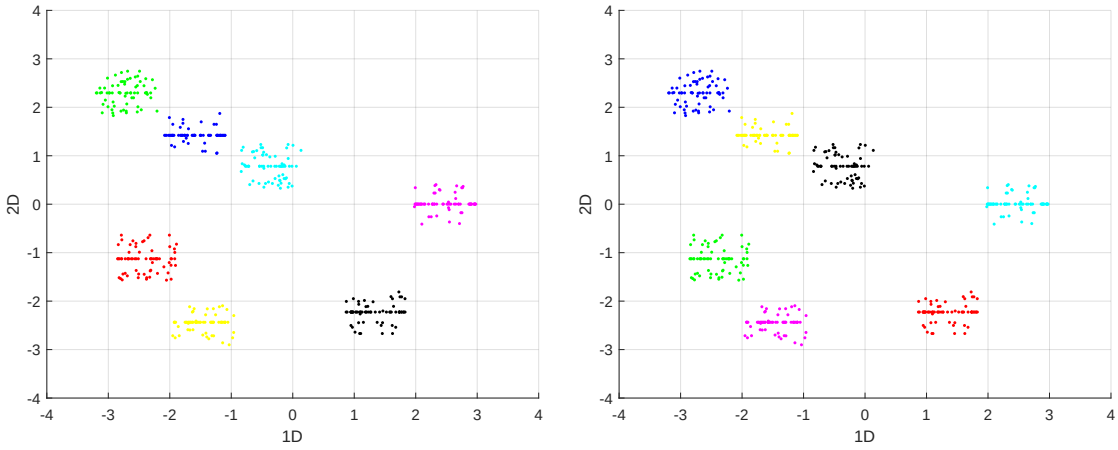
Lastly, there is a pattern that if a certain-layered neural network has highest accumulated rewards, then it tends to also get highest accuracy. For example, at the right column on Figure 6.18, for all algorithms, yellow line is on the top of the other colors, which means 3-layered neural network accumulated reward greater than the others. Along this fact, at the right column on Figure 6.4, yellow lines of all algorithms also shows high accuracy performance. This can be more supported by looking in Table 6.3 where 3-layered neural network of each algorithms shows high performance.

6.2.4 Dynamics Analysis

The fourth measurement of CALM performance is dynamic analysis. Dynamic analysis is to check how CALM algorithms are able to adapt to dynamically changed environment. In order to provide dynamic changes in synthetic data, DATA6 is used as shown in Figure 6.27. DATA6 includes two data subsets: (1) 7 data clusters each with 70 data points (in total 490 data points) and (2) the same 7 data clusters each with 70 data points (in total 490 data points) in which each data point has the same input value but different target output values from the first data subset. Therefore, the total number of data points in DATA6 is 980 and each data subset has half of them with different target output for each cluster. Note that the first data subset is named *unchanged data subset* of DATA6 as shown in Figure 6.27a and the second data subset is named *changed data subset* of DATA6 as shown in Figure 6.27b.

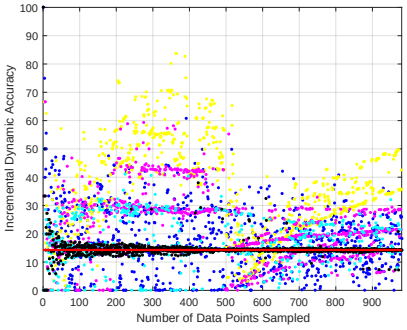
Figure 6.27a shows each data cluster has different target output and the 7 target outputs are represented with each different corresponding color: green-OUTPUT1, blue-OUTPUT2, yellow-OUTPUT3, magenta-OUTPUT4, cyan-OUTPUT5, black-OUTPUT6, red-OUTPUT7. Figure 6.27b shows how each clustered target output is switched to representing dynamical environment changes. This means after 490 learning steps a learning system will have changed target output: OUTPUT1 (green) to OUTPUT2 (blue), OUTPUT2 to OUTPUT3 (yellow), OUTPUT3 to OUTPUT4 (magenta), OUTPUT4 to OUTPUT5 (cyan), OUTPUT5 to OUTPUT6 (black), OUTPUT6 to OUTPUT7 (red), and OUTPUT7 to OUTPUT1 (green). Note that each context in first and second data subset has same Cartesian value but only has different target output so as to provide that the contexts learned earlier are no longer correct after the

learning step 490.

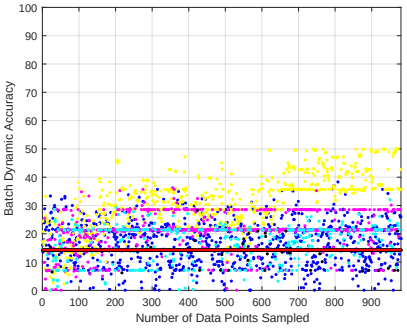


(a) Data6 Before Dynamic Changes (b) Data6 After Dynamic Changes

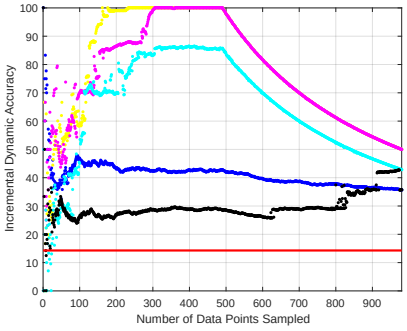
Figure 6.27: Dynamic Data Sets



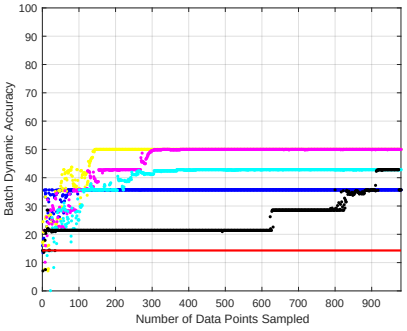
(a) CALM-rLRB ($ITR = 200$)



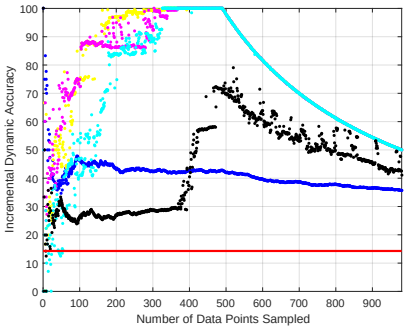
(b) CALM-rLRB ($ITR = 200$)



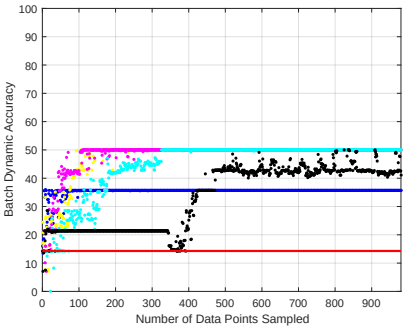
(c) CALM-eLRB ($ITR = 200$)



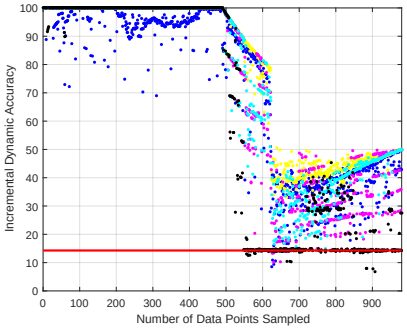
(d) CALM-eLRB ($ITR = 200$)



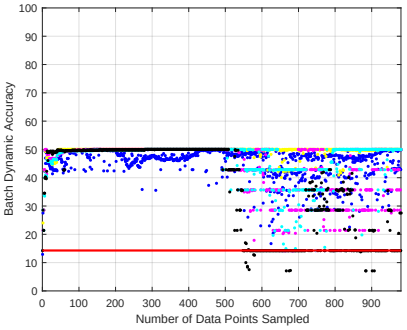
(e) CALM-epLRB ($ITR = 200$)



(f) CALM-epLRB ($ITR = 200$)

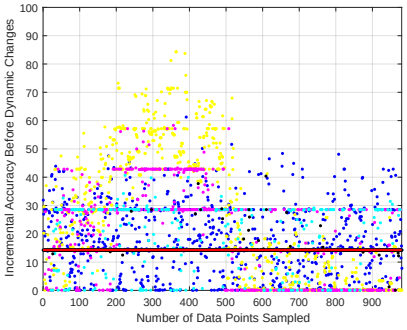


(g) CALM-nepLRB ($ITR = 200$)

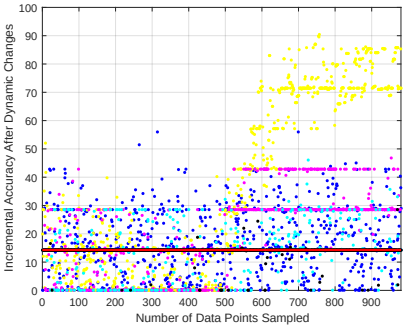


(h) CALM-nepLRB ($ITR = 200$)

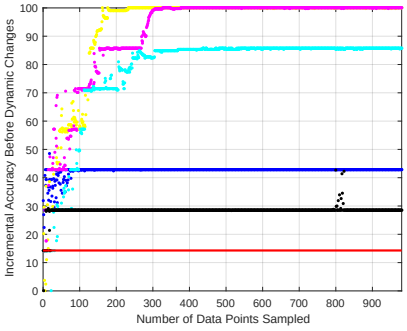
Figure 6.28: Dynamic Accuracy on Data 6



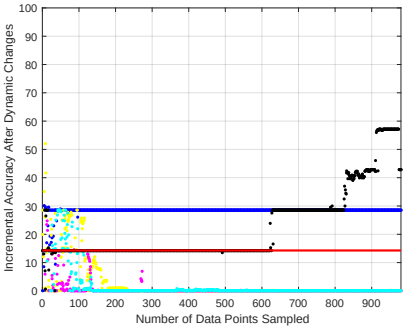
(a) CALM-rLRB ($ITR = 200$)



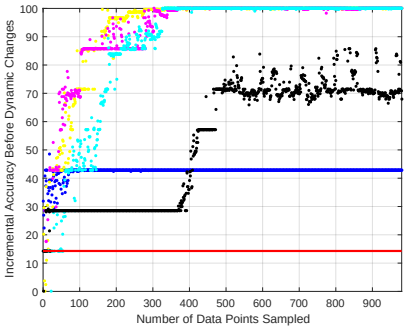
(b) CALM-rLRB ($ITR = 200$)



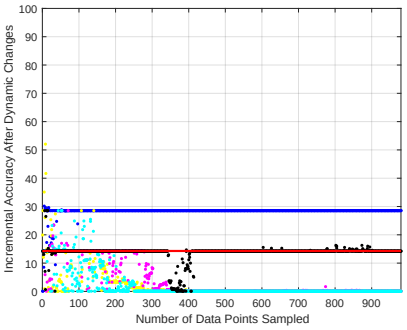
(c) CALM-eLRB ($ITR = 200$)



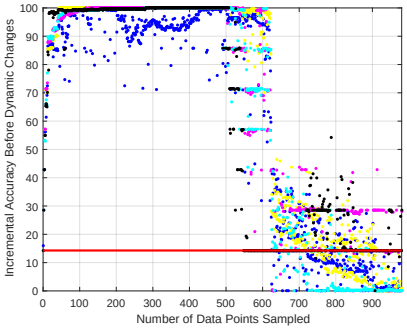
(d) CALM-eLRB ($ITR = 200$)



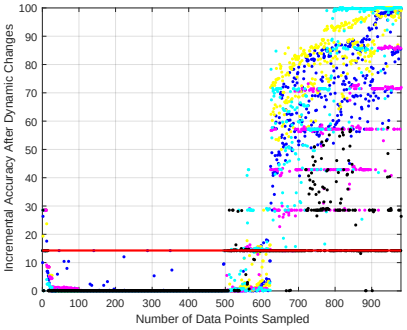
(e) CALM-epLRB ($ITR = 200$)



(f) CALM-epLRB ($ITR = 200$)



(g) CALM-nepLRB ($ITR = 200$)



(h) CALM-nepLRB ($ITR = 200$)

Figure 6.29: Dynamic Accuracy on Data 6 Comparison

In dynamic analysis, two kinds of evaluation methods are applied: (1) accuracy and (2) EKB status transition.

The first evaluation method in dynamic analysis is checking dynamic accuracy. Four ways of checking accuracy are used for each CALM algorithm with five depths of CALM-ANNs: (1) incremental dynamic accuracy, (2) batch dynamic accuracy, (3) incremental dynamic accuracy only on unchanged data subset, and (4) incremental dynamic accuracy only on changed data subset. Having these four ways of checking accuracy aims to have different perspectives of analyzing the results and to provide accuracy validation; With having the ways, we can have more clear understanding of CALM performance results and eventually we can see each result supports each others. In this regard, note that the DATA6 itself is not divided into training and testing data and thus 5-fold cross-validation is not applied in the experiments on DATA6. The accuracy validation is evaluated in subsection 6.2.1 thus in this subsection checking accuracy is focused on looking into the process of responding to changed environment by measuring four different ways of accuracy.

The second evaluation method in dynamic analysis is checking EKB status transition. In this method, the experiences stored in KEB at several learning steps are plotted in order to check how CALM can change the stored experiences during learning process when an environment changes dynamically.

Figure 6.28 shows incremental accuracy (left column) and batch accuracy (right column) of each algorithm with five depths of CALM-ANNs. Incremental accuracy is measured in each learning step by calculating percentage of the number of successfully classified contexts on DATA6 in a incremental way. This is exactly same way of measuring training accuracy on from DATA1 to

DATA5 in subsection 6.2.1. Note that the maximum accuracy can not be over 50% at the end of learning since DATA6 has two data subsets where each cluster has switched to different target output after half of learning the steps, 490.

First, all of CALM algorithms show increasing accuracy up to around learning step 490 and start to diminish after that. Decreased accuracy after learning step 490 is expected since the second data subset of DATA6 is used after that learning step.

In CALM-rLRB as shown in Figure 6.28a, CALM-ANN2 (yellow) shows best accuracy among the other CALM-ANNs. CALM-ANN2 mostly shows highest accuracy at overall learning steps. Also, it starts to learn the changed environment after learning step 490 and shows highest accuracy at the end of learning compared to the other algorithms. This means CALM-ANN2, which is 3-layered neural network, is most effective neural network for CALM-rLRB on DATA6.

In CALM-eLRB as shown in Figure 6.28c, CALM-ANN2 and CALM-ANN3 reaches 100% accuracy before the half learning step and shows gradual decreasing accuracy for the rest of learning period; also CALM-ANN2 reaches full accuracy before CALM-ANN3 reaches. CALM-ANN4 shows similar pattern in dynamic changes but less increasing accuracy and more decreasing accuracy compared to CALM-ANN2 and CALM-ANN3. CALM-ANN1 and CALM-ANN6 have no learning effects on understanding both unchanged and changed environment. In this regard, CALM-ANN2 shows best performance for CALM-eLRB on DATA6.

The big difference of CALM-eLRB dynamic accuracy from CALM-rLRB is that CALM-rLRB shows sporadic accuracy distribution while CALM-eLRB shows monotonic pattern. This shows how experience-based learning reward

learning react to an environment differently. CALM-eLRB uses accumulated past successive experiences thus when the data is changed it gradually get confused. On the other hand, CALM-rLRB uses only currently given context in optimizing its neural network therefore it is affected more dramatically when an environment changed.

In CALM-epLRB as shown in Figure 6.28e, CALM-ANN2, CALM-ANN3, and CALM-ANN4 shows excellent performance with reaching 100% accuracy before dynamic change and shows gradual decrease down to 50% when an environment is changed, which means all the three algorithms can perfectly understand at least the unchanged environment. CALM-ANN5 starts to show its learning effects around learning step 400% and shows gradual decrease when it meets changed environment. CALM-ANN1 has no learning effects. In this regard, it is considered that CALM-epLRB shows better performance than CALM-eLRB on DATA6.

In CALM-nepLRB as shown in Figure 6.28g, all of CALM-nepLRB-ANNs shows high accuracy before dynamic changes and shows sporadic decreases after dynamic changes occur. This is because CALM-nepLRB uses a novel neural networks, CALM-nepLRB-ANNs, and it selects its algorithm in a flexible way as described in Chapter 5.5. Also, we can see all of CALM-nepLRB-ANNs try to learn the changed environment. CALM-ANN2, CALM-ANN3, and CALM-ANN4 shows 50% accuracy at the end of learning.

Batch accuracy is measured in each learning step by applying currently learned CALM-ANN onto whole data set which has 980 data points. The goal of this is to check how much a system can adapt to an environment where each

context has two conflicting target output values. Therefore, this accuracy is more like testing CALM in an inconsistent environment. Note that batch accuracy results at the end of learning should be exactly same as the incremental accuracy results; this is because each algorithm eventually checking its both incremental and bath accuracy on the whole number of data points at the last learning step.

In CALM-rLRB as shown in Figure 6.28b, we can see CALM-ANN2 tries to understand inconsistent environment with highest accuracy overall while the other CALM-ANNs get confused with showing sporadic accuracy distribution. In CALM-eLRB as shown in Figure 6.28d, CALM-ANN2 and CALM-ANN3 reaches 50% accuracy at early learning step, which is maximum accuracy on the fully inconsistent environment, and stayed for the rest of learning period; this is exactly same as the results of incremental learning. In CALM-epLRB as shown in Figure 6.28f, CALM-ANN2, CALM-ANN3, and CALM-ANN4 shows excellent performance with reaching 50% accuracy which also supports the results of incremental accuracy. Like that in the batch dynamic accuracy, CALM-ANN4 and CALM-ANN5 shows better performance in CALM-epLRB than CALM-eLRB; also, CALM-ANN2 and CALM-ANN3 of CALM-epLRB reaches the highest accuracy in earlier learning steps than in CALM-eLRB. This means the Selective-Power-Update rule had impact on improving the learning results by powering the use of experiences on DATA6. In CALM-nepLRB as shown in Figure 6.28h, before dynamic changes most CALM-nepLRB-ANNs shows 50% accuracy at the very early learning step and tries to understand dynamically changed environment, which also supports the results of incremental accuracy.

Figure 6.29 shows incremental dynamic accuracy on both unchanged and

changed data; the left column on the figure represents incremental accuracies of each algorithms with each CALM-ANN which used only unchanged data set. On the other hand, right column shows incremental dynamic accuracies of each algorithm with each CALM-ANN which is tested only on changed data set. More specifically, incremental accuracy before dynamic changes is measured in each learning step by applying currently learned CALM-ANN onto the first data subset of DATA6 which has 490 data points. Incremental accuracy after dynamic changes is measured in each learning step by applying currently learned CALM-ANN onto the second data subset of DATA6 which has also 490 data points. The goal of checking the dynamic accuracy by applying two separate data subsets is to see how each learned CALM-ANN in each algorithm of CALM can represent the unchanged environment and changed environment respectively.

CALM-rLRB and CALM-nepLRB are dynamic algorithms. CALM-rLRB and CALM-nepLRB shows decreased accuracy when environment is changed on unchanged data of DATA6 as shown Figure 6.29a and Figure 6.29g; on the other hand, they shows increased accuracy on changed data subset after environment is changed as shown Figure 6.29b and Figure 6.29h. This implies CALM-rLRB and CALM-nepLRB learn first unchanged environment and then learn the changed environment.

In Figure 6.29a, CALM-rLRB, especially CALM-ANN2, shows high accuracy when it is tested its accuracy on unchanged data but gives low accuracy when it applied on changed data. This means CALM-rLRB started to changed new environment so it can not give high accuracy on the unchanged old data.

In Figure 6.29b, CALM-rLRB shows low accuracy when it is tested on unchanged data but gives high accuracy when it is applied on changed data subset. This means CALM-rLRB could learn the unchanged environment so it

cannot give high accuracy on changed environment but could give high accuracy after learning step 490.

In Figure 6.29g, CALM-nepLRB shows mostly highest accuracy on unchanged data but the accuracy decreases after learning step around 600 on unchanged data. This means CALM-nepLRB could successfully learn the changed environment so it could not show high accuracy on unchanged data set.

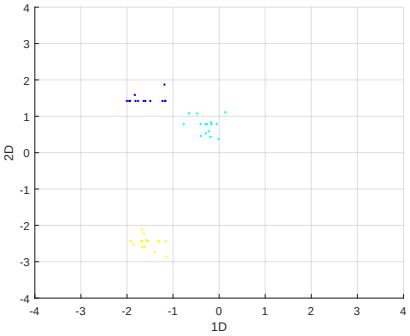
In Figure 6.29h, CALM-nepLRB shows low accuracy on changed data subset but show high accuracy after the learning step around 600, which menas CALM-nepLRB could learn the changed environment.

It is notable that CALM-nepLRB shows dropping its accuracy and increasing its accuracy around the learning step 600 which is not 490. This implies that CALM-nepLRB can remember the previous environment in a longer period than CALM-rLRB. It is also notable that CALM-nepLRB can survive in dynamic environment with very high accuracy, especially on DATA6. Especially in unchanged environment, except for CALM-ANN1, all of CALM-nepLRB shows 100% accuracy at the end of the unchanged environment, 490. In changed environment, CALM-ANN2, CALM-ANN3, and CALM-ANN4 show 100% accuracy while CALM-ANN1 shows around 85% and CALM-ANN5 shows around 25%. In CALM-nepLRB, even shallow learning shows 85% accuracy.

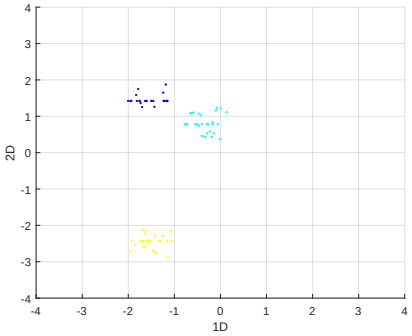
CALM-eLRB and CALM-epLRB are not dynamic algorithms. CALM-eLRB and CALM-epLRB show increasing accuracy before environment changes and stays the increase accuracy even after the environment changed on unchanged data subset as shown Figure 6.29c and Figure 6.29e. On the other hand, CALM-eLRB and CALM-epLRB get confused on changed data subset before the environment changes and then shows no learning effects after the environment changes on changed data subset. This means CALM-eLRB and CALM-epLRB

could successfully learn the unchanged environment but could not adapt to newly changed environment once it learned a certain environment.

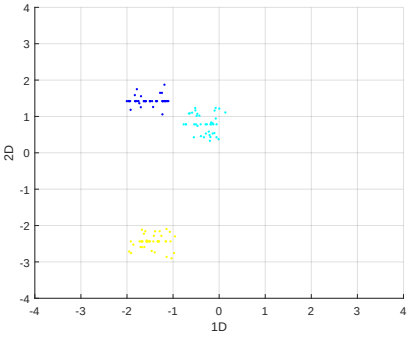
From these experiments, we can conclude the following things. CALM-rLRB shows lowest accuracy in static environment compared to CALM-eLRB and CALM-epLRB but can adapt to dynamic environment by re-learning the new environment. CALM-nepLRB shows highest accuracy and also can survive in dynamic environment.



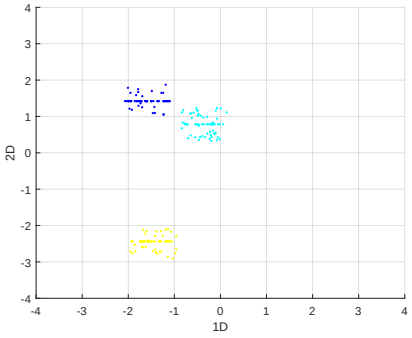
(a) $LearningStep = 100$



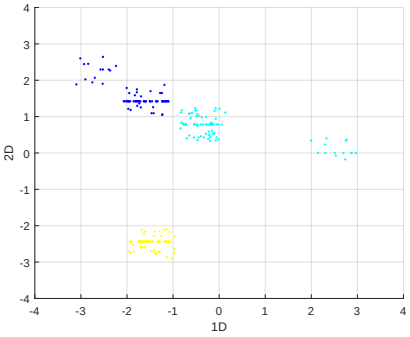
(b) $LearningStep = 200$



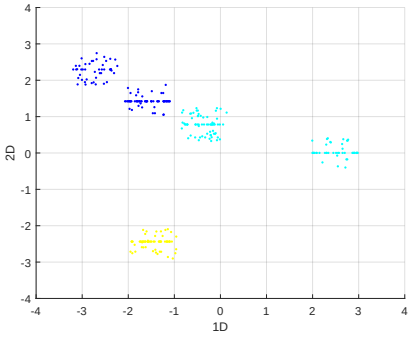
(c) $LearningStep = 300$



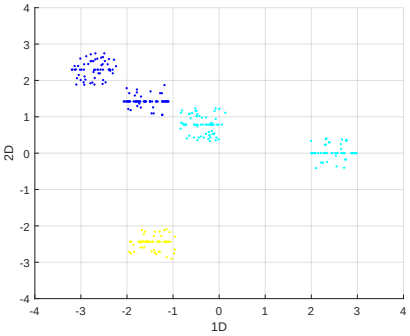
(d) $LearningStep = 500$



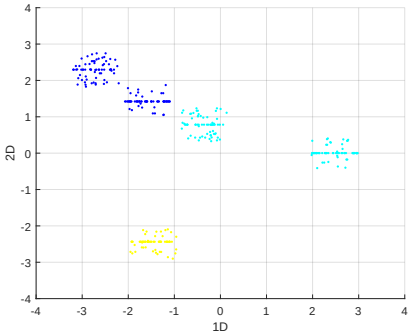
(e) $LearningStep = 600$



(f) $LearningStep = 800$

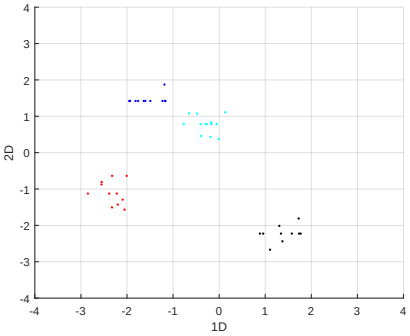


(g) $LearningStep = 900$

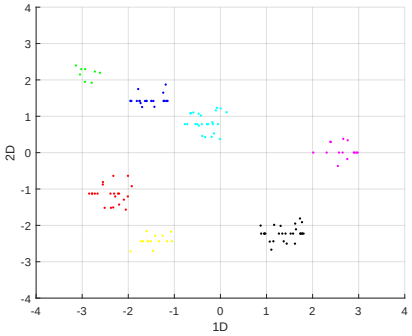


(h) $LearningStep = 980$

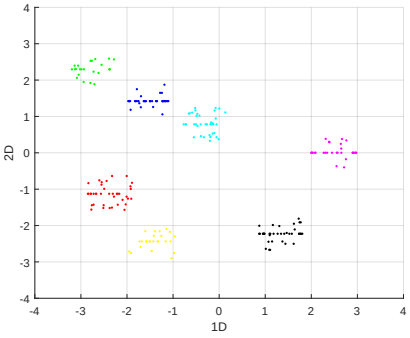
Figure 6.30: CALM-eLRB EKB Transition on Data 6 ($L = 2$)



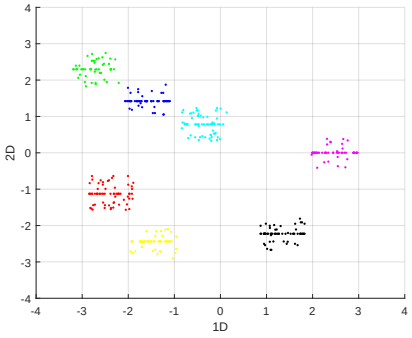
(a) $LearningStep = 100$



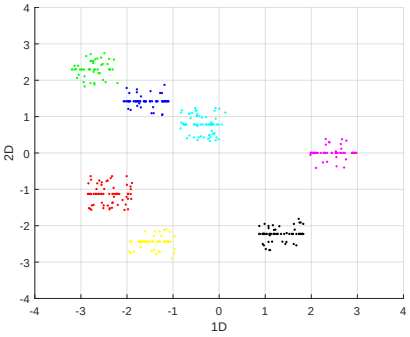
(b) $LearningStep = 200$



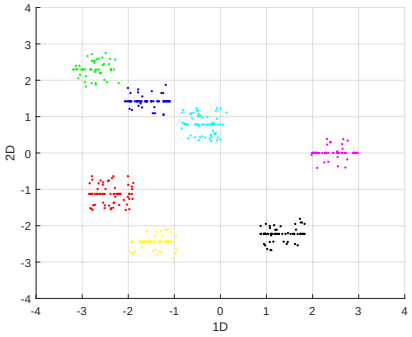
(c) $LearningStep = 300$



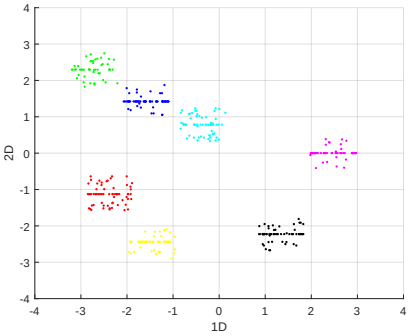
(d) $LearningStep = 500$



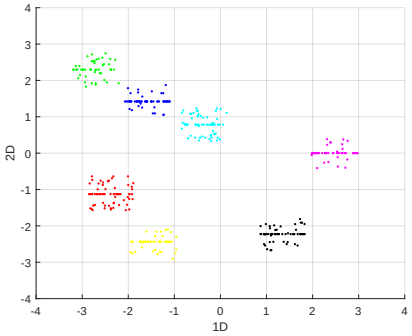
(e) $LearningStep = 600$



(f) $LearningStep = 800$

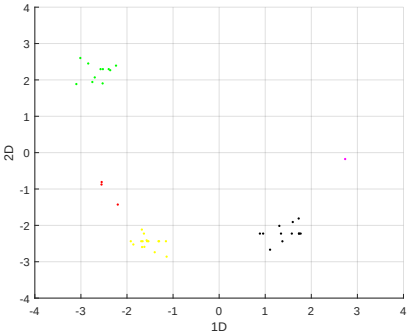


(g) $LearningStep = 900$

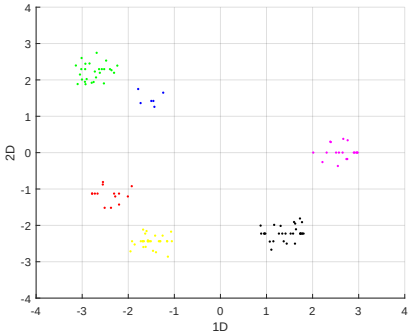


(h) $LearningStep = 980$

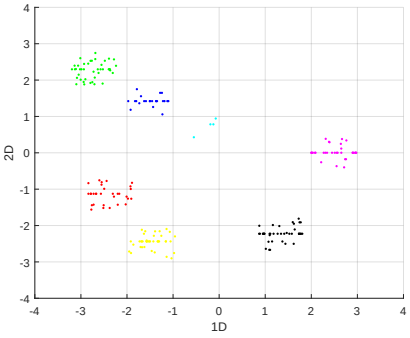
Figure 6.31: CALM-eLRB EKB Transition on Data 6 ($L = 3$)



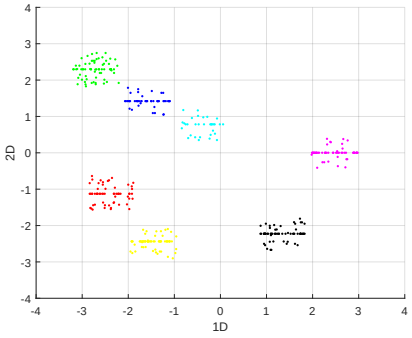
(a) $LearningStep = 100$



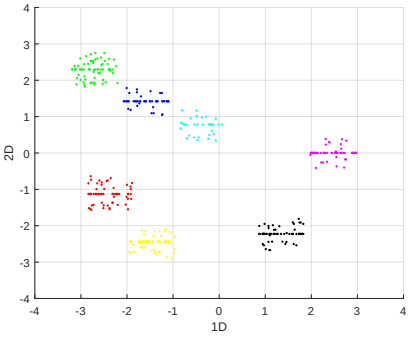
(b) $LearningStep = 200$



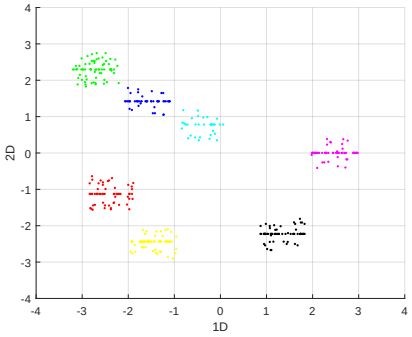
(c) $LearningStep = 300$



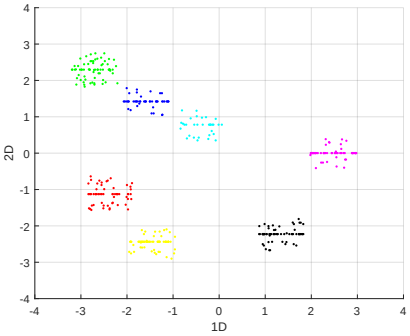
(d) $LearningStep = 500$



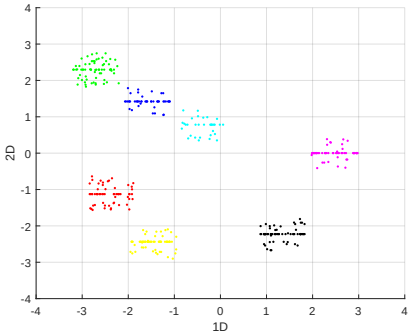
(e) $LearningStep = 600$



(f) $LearningStep = 800$

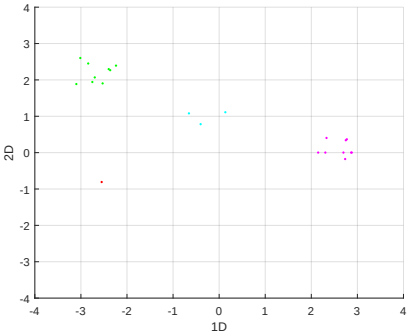


(g) $LearningStep = 900$

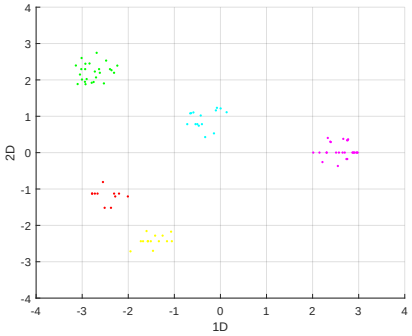


(h) $LearningStep = 980$

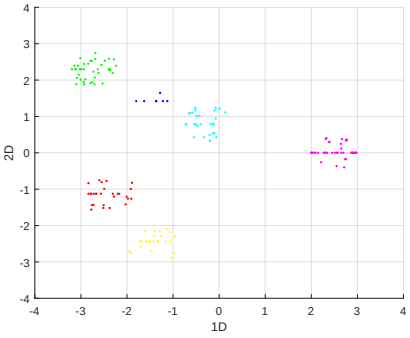
Figure 6.32: CALM-eLRB EKB Transition on Data 6 ($L = 4$)



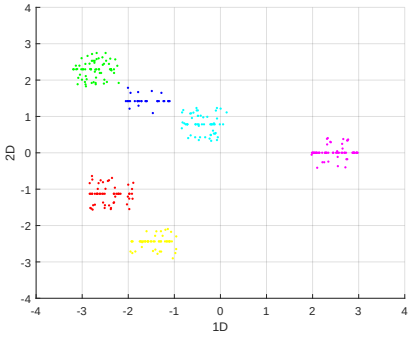
(a) $LearningStep = 100$



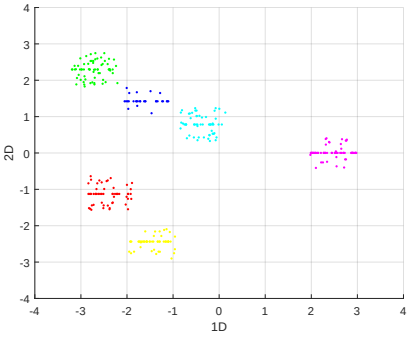
(b) $LearningStep = 200$



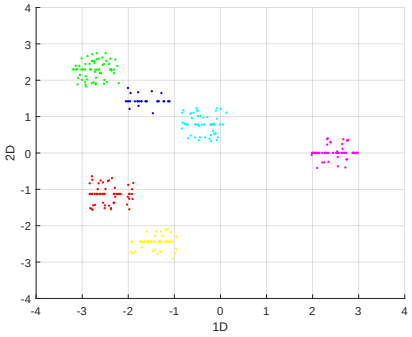
(c) $LearningStep = 300$



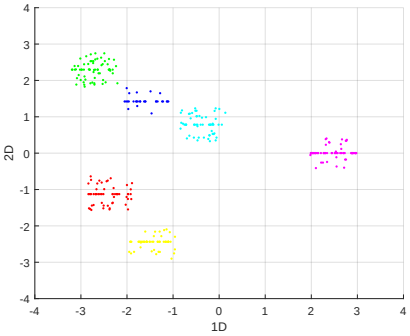
(d) $LearningStep = 500$



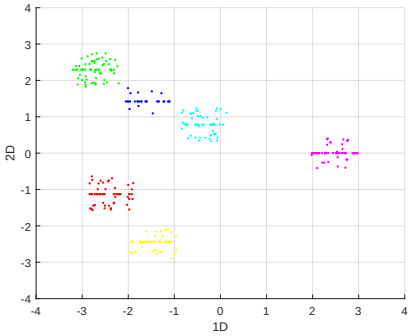
(e) $LearningStep = 600$



(f) $LearningStep = 800$

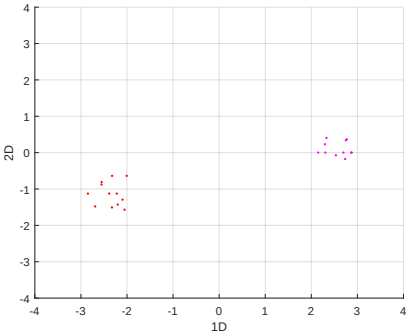


(g) $LearningStep = 900$

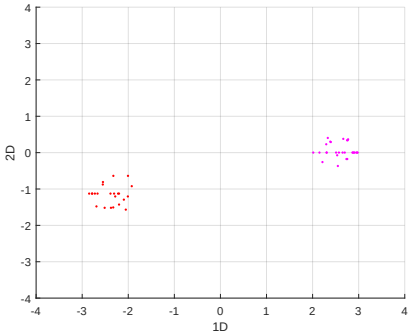


(h) $LearningStep = 980$

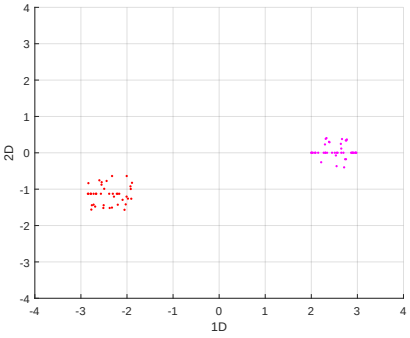
Figure 6.33: CALM-eLRB EKB Transition on Data 6 ($L = 5$)



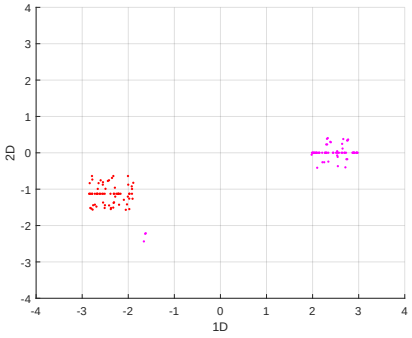
(a) $LearningStep = 100$



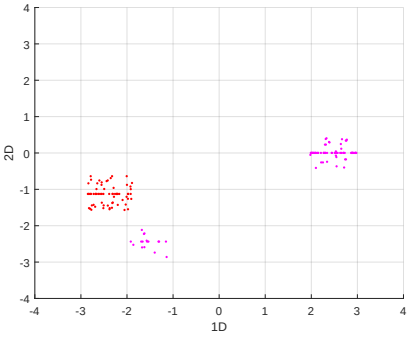
(b) $LearningStep = 200$



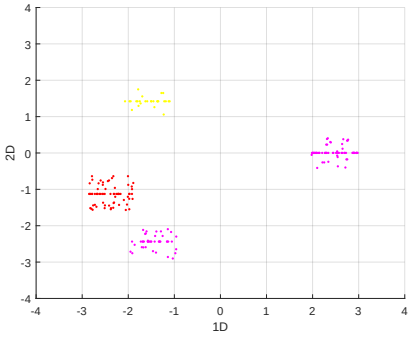
(c) $LearningStep = 300$



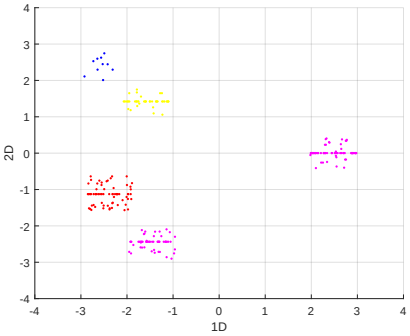
(d) $LearningStep = 500$



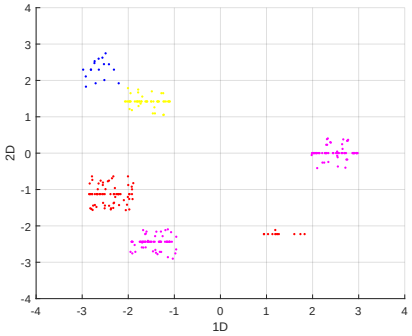
(e) $LearningStep = 600$



(f) $LearningStep = 800$

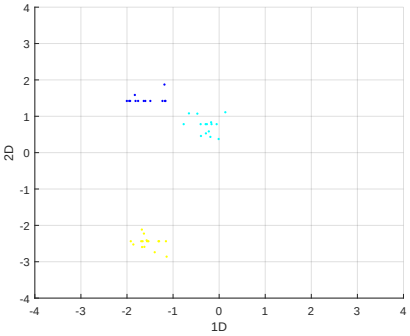


(g) $LearningStep = 900$

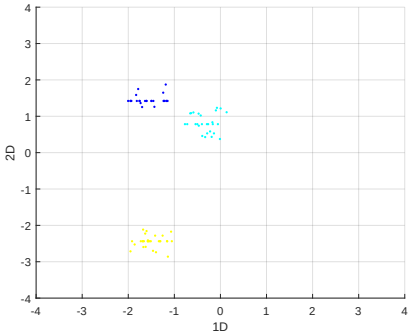


(h) $LearningStep = 980$

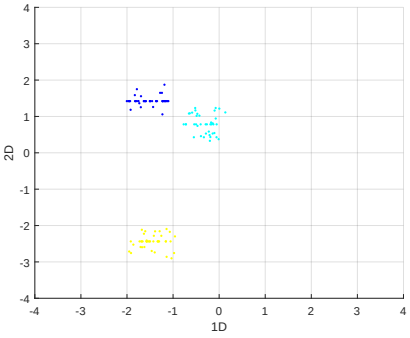
Figure 6.34: CALM-eLRB EKB Transition on Data 6 ($L = 6$)



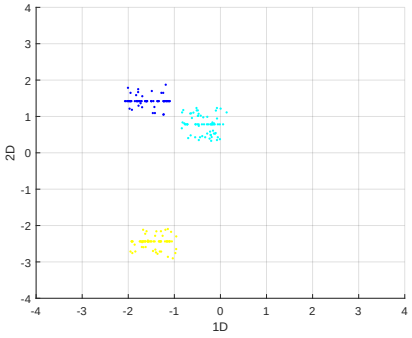
(a) $LearningStep = 100$



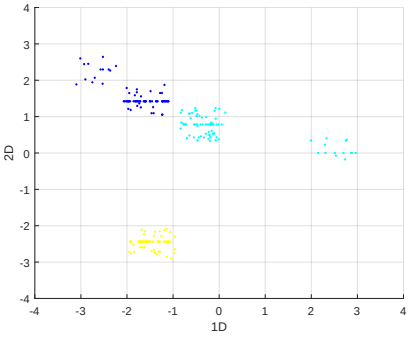
(b) $LearningStep = 200$



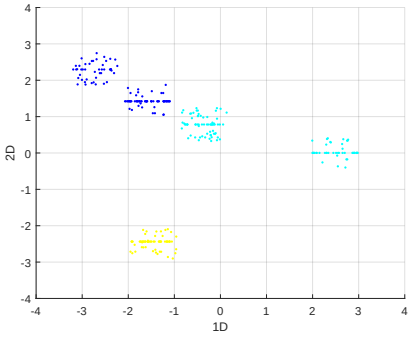
(c) $LearningStep = 300$



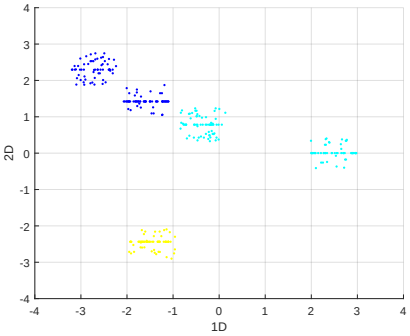
(d) $LearningStep = 500$



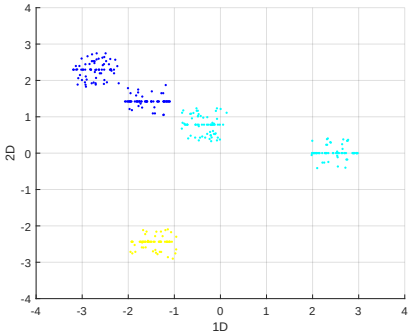
(e) $LearningStep = 600$



(f) $LearningStep = 800$

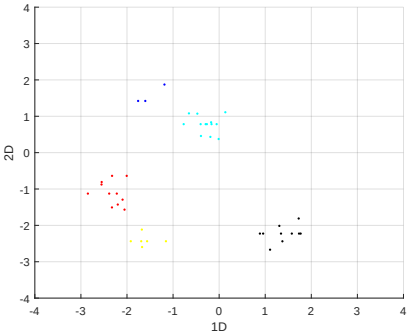


(g) $LearningStep = 900$

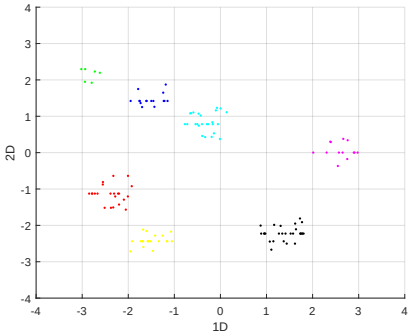


(h) $LearningStep = 980$

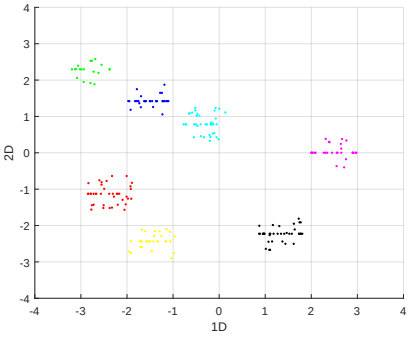
Figure 6.35: CALM-epLRB EKB Transition on Data 6 ($L = 2$)



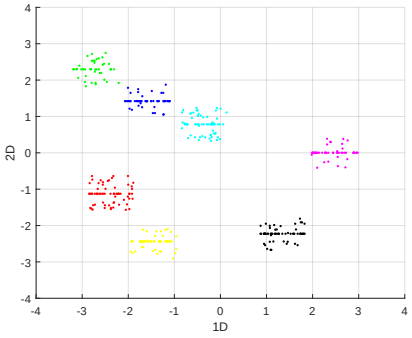
(a) $LearningStep = 100$



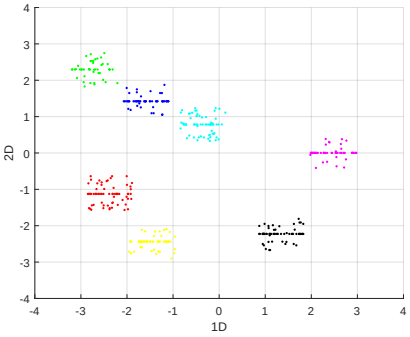
(b) $LearningStep = 200$



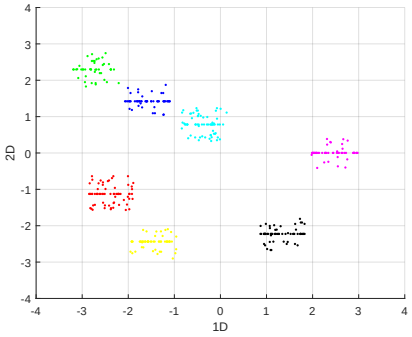
(c) $LearningStep = 300$



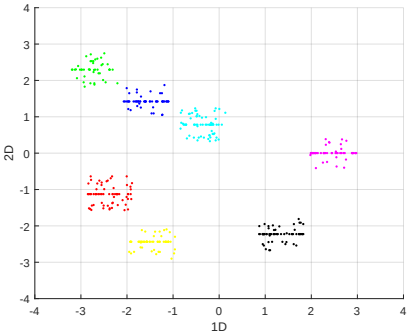
(d) $LearningStep = 500$



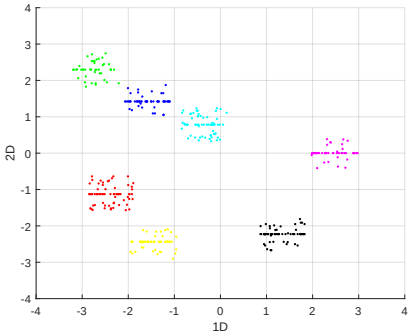
(e) $LearningStep = 600$



(f) $LearningStep = 800$

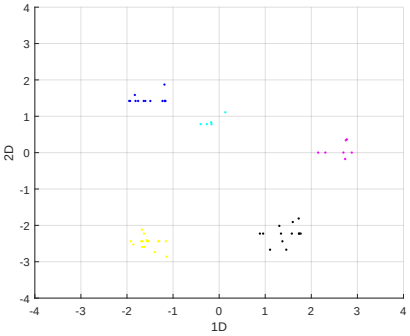


(g) $LearningStep = 900$

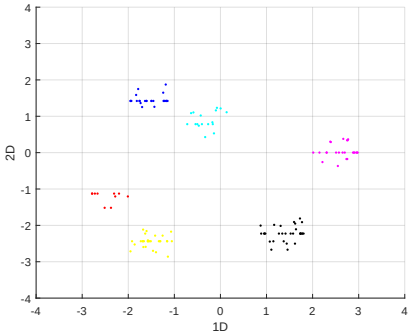


(h) $LearningStep = 980$

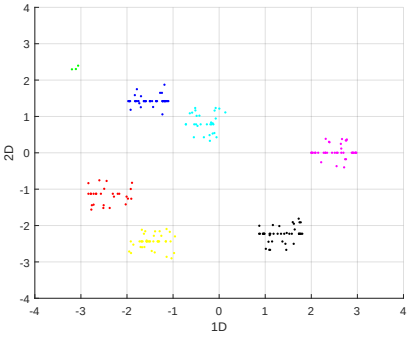
Figure 6.36: CALM-epLRB EKB Transition on Data 6 ($L = 3$)



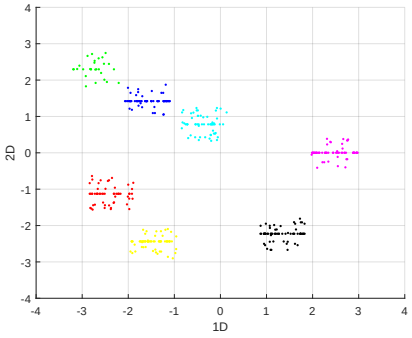
(a) $LearningStep = 100$



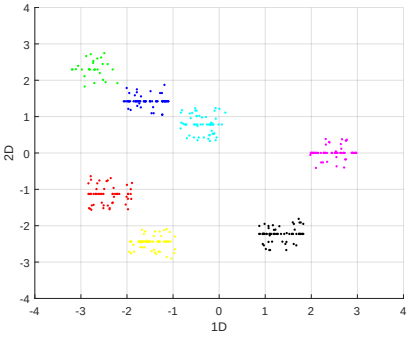
(b) $LearningStep = 200$



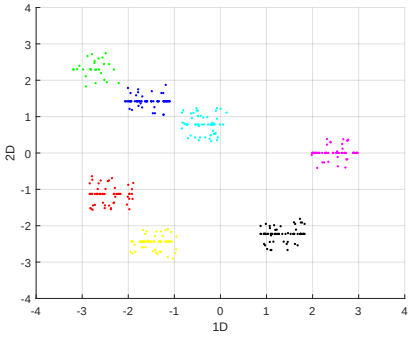
(c) $LearningStep = 300$



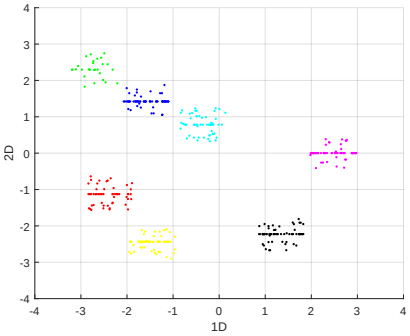
(d) $LearningStep = 500$



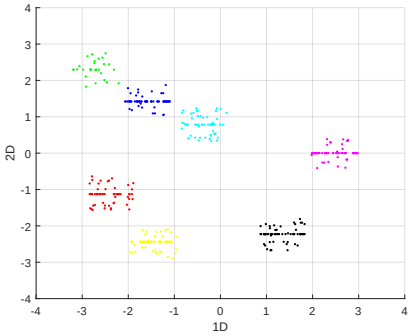
(e) $LearningStep = 600$



(f) $LearningStep = 800$

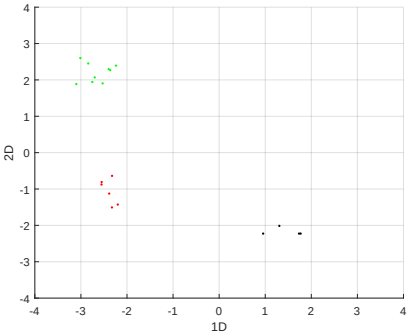


(g) $LearningStep = 900$

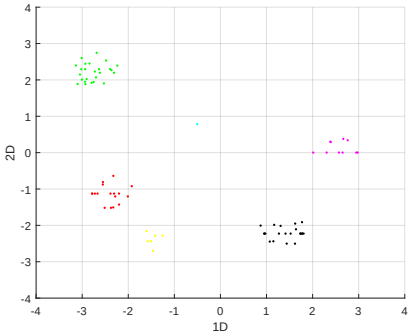


(h) $LearningStep = 980$

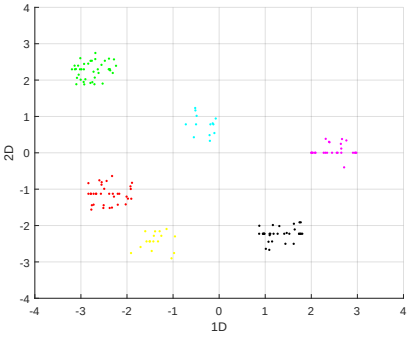
Figure 6.37: CALM-epLRB EKB Transition on Data 6 ($L = 4$)



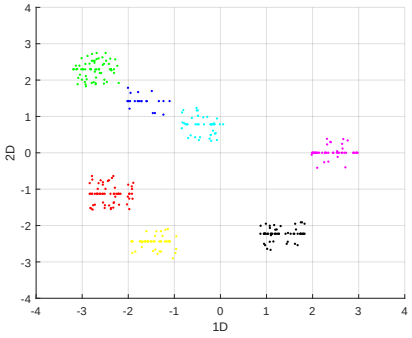
(a) $LearningStep = 100$



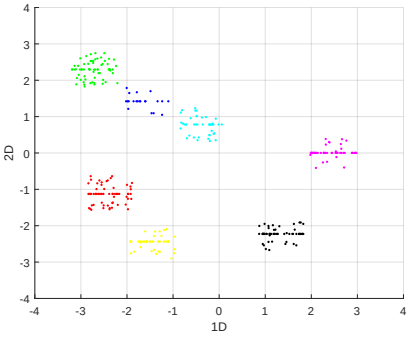
(b) $LearningStep = 200$



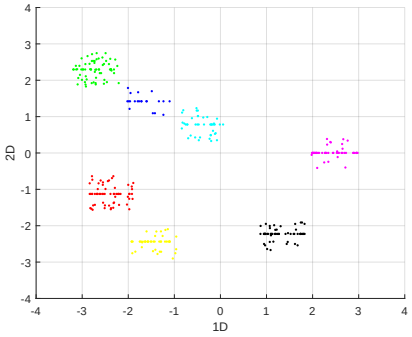
(c) $LearningStep = 300$



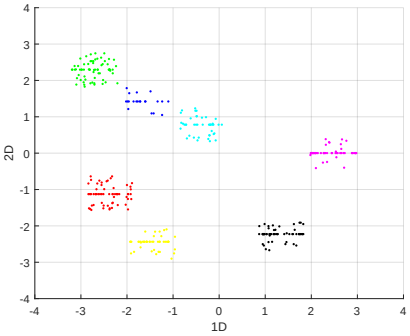
(d) $LearningStep = 500$



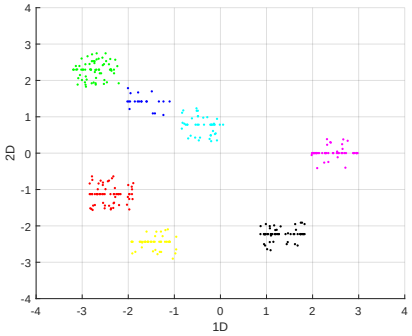
(e) $LearningStep = 600$



(f) $LearningStep = 800$

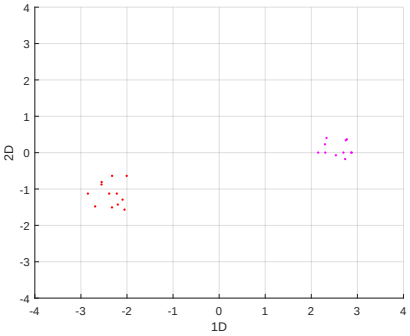


(g) $LearningStep = 900$

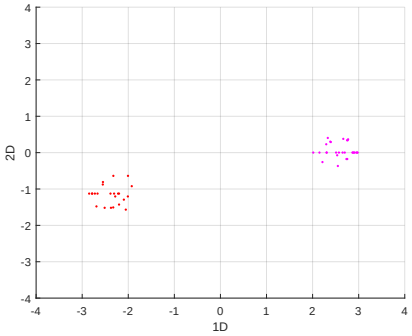


(h) $LearningStep = 980$

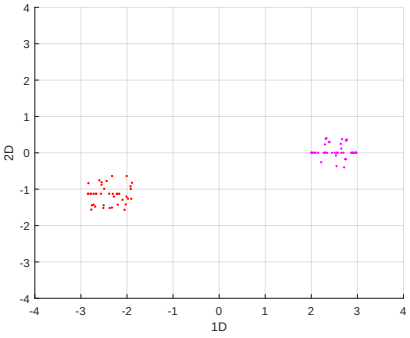
Figure 6.38: CALM-epLRB EKB Transition on Data 6 ($L = 5$)



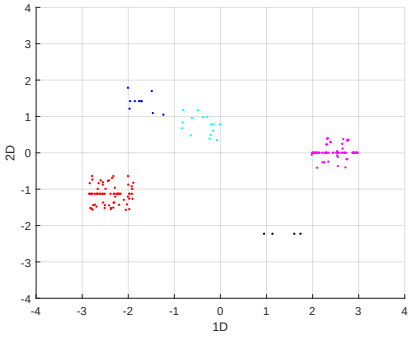
(a) $LearningStep = 100$



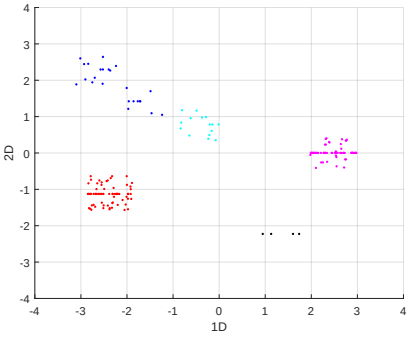
(b) $LearningStep = 200$



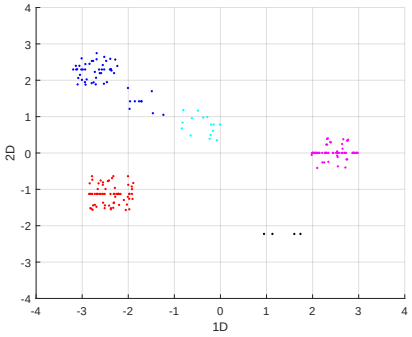
(c) $LearningStep = 300$



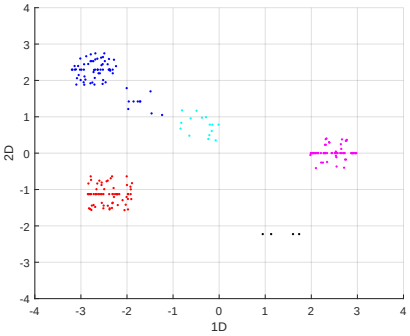
(d) $LearningStep = 500$



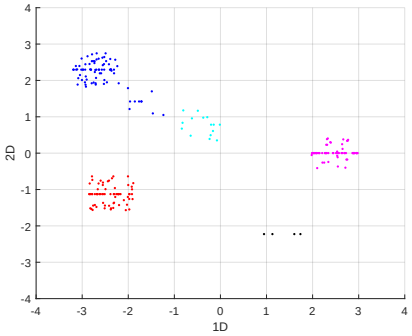
(e) $LearningStep = 600$



(f) $LearningStep = 800$

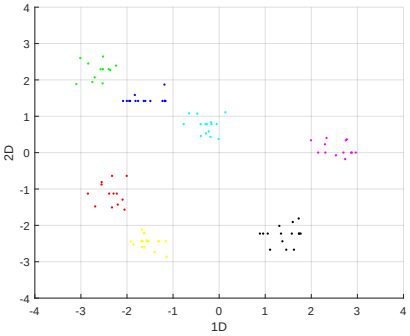


(g) $LearningStep = 900$

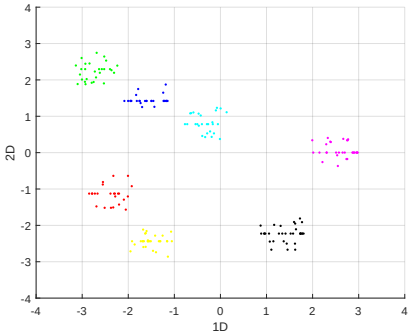


(h) $LearningStep = 980$

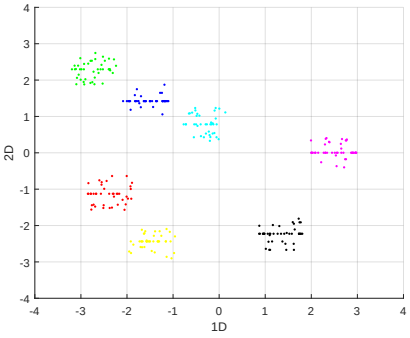
Figure 6.39: CALM-epLRB EKB Transition on Data 6 ($L = 6$)



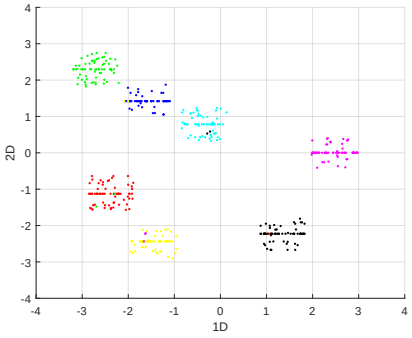
(a) $LearningStep = 100$



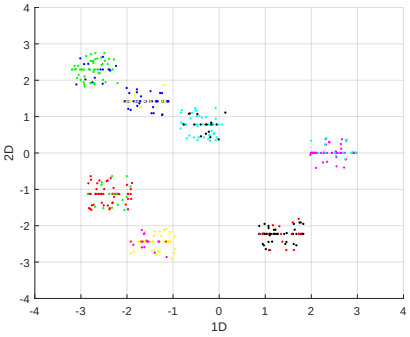
(b) $LearningStep = 200$



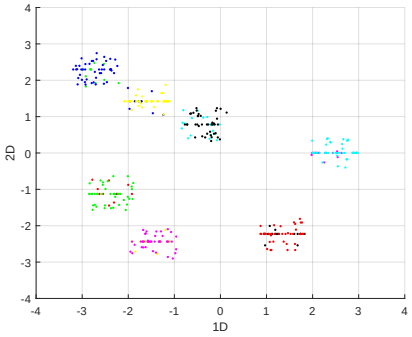
(c) $LearningStep = 300$



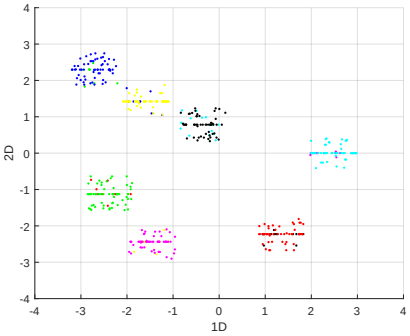
(d) $LearningStep = 500$



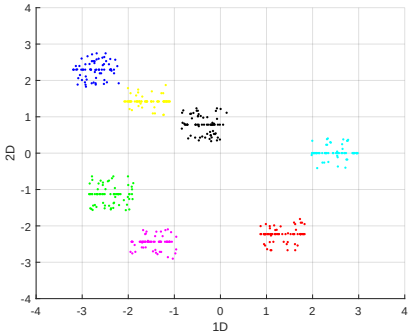
(e) $LearningStep = 600$



(f) $LearningStep = 800$

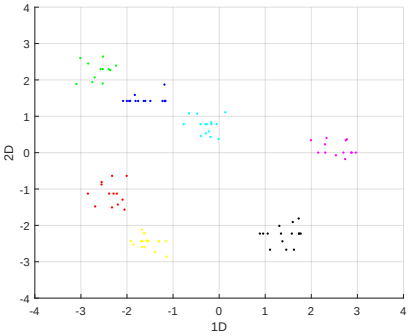


(g) $LearningStep = 900$

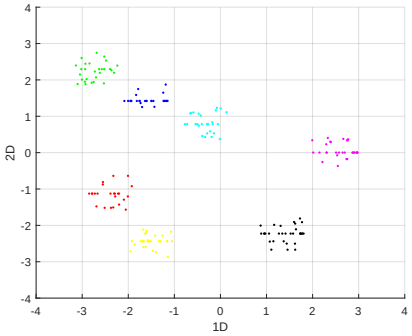


(h) $LearningStep = 980$

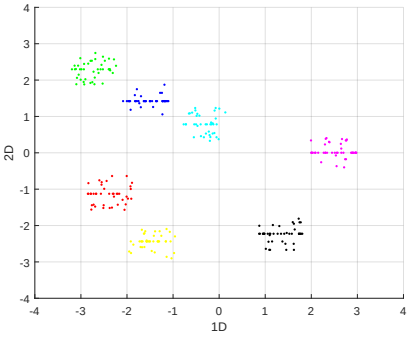
Figure 6.40: CALM-nepLRB EKB Transition on Data 6 ($L = 2$)



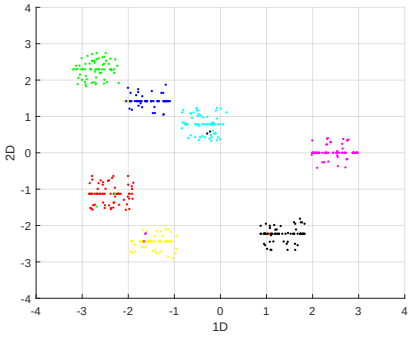
(a) $LearningStep = 100$



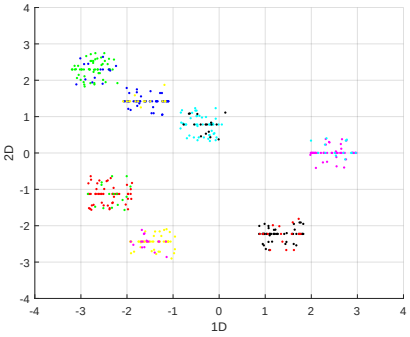
(b) $LearningStep = 200$



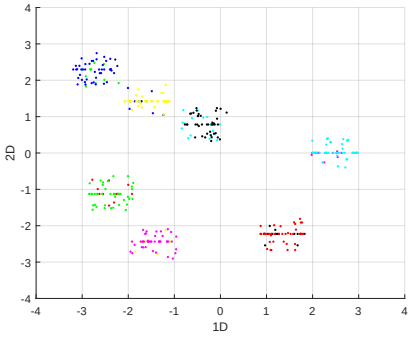
(c) $LearningStep = 300$



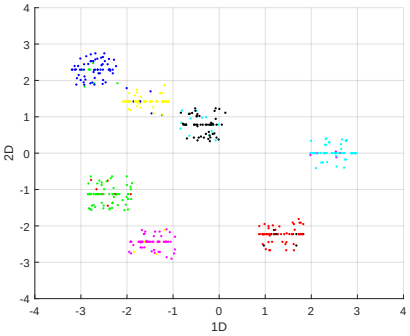
(d) $LearningStep = 500$



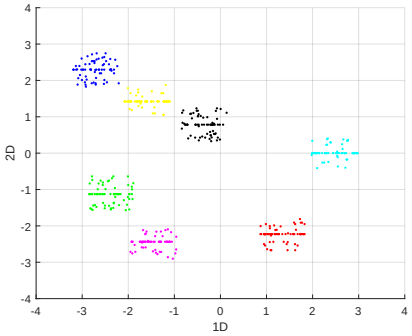
(e) $LearningStep = 600$



(f) $LearningStep = 800$

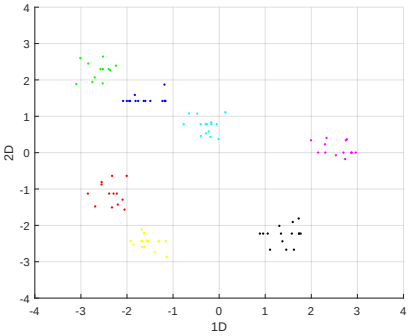


(g) $LearningStep = 900$

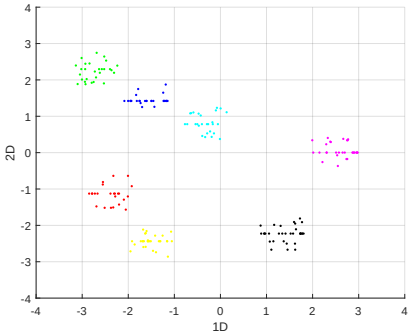


(h) $LearningStep = 980$

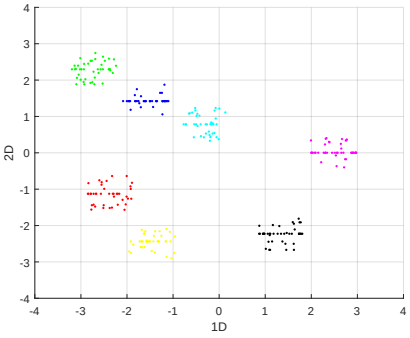
Figure 6.41: CALM-nepLRB EKB Transition on Data 6 ($L = 3$)



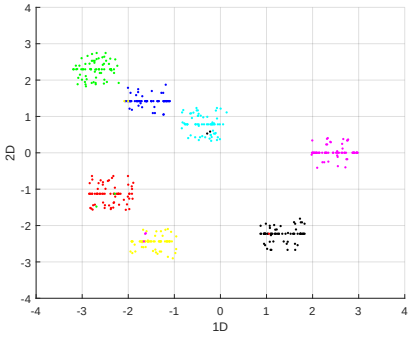
(a) $LearningStep = 100$



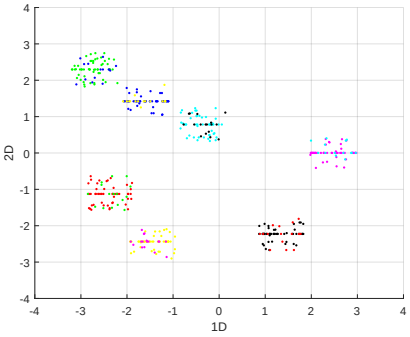
(b) $LearningStep = 200$



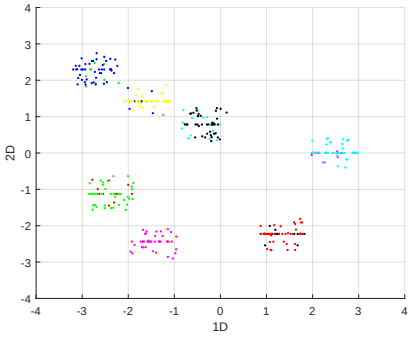
(c) $LearningStep = 300$



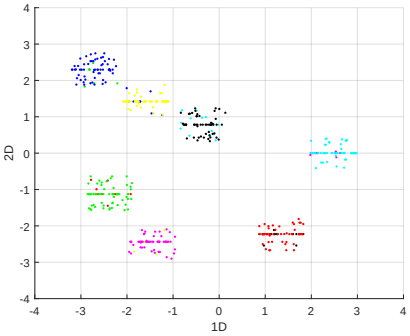
(d) $LearningStep = 500$



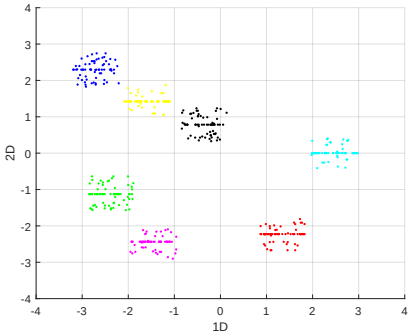
(e) $LearningStep = 600$



(f) $LearningStep = 800$

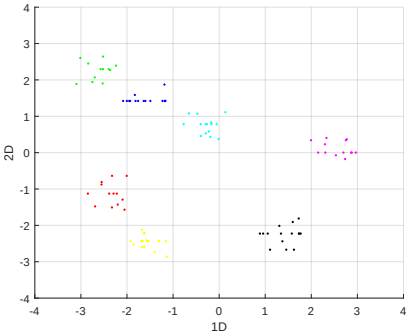


(g) $LearningStep = 900$

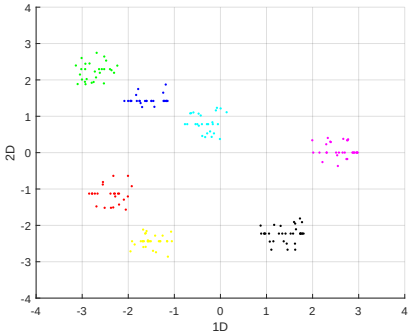


(h) $LearningStep = 980$

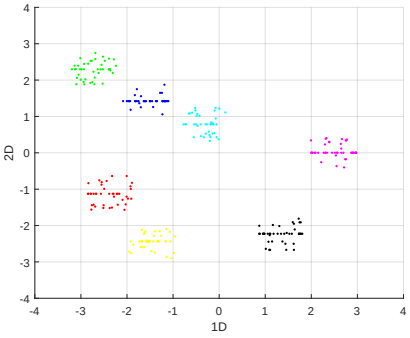
Figure 6.42: CALM-nepLRB EKB Transition on Data 6 ($L = 4$)



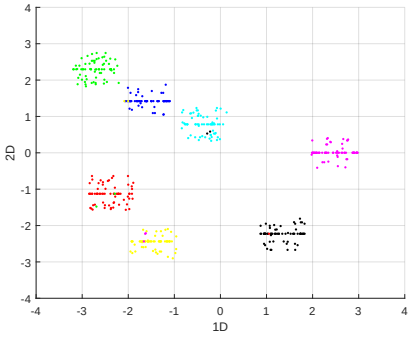
(a) $LearningStep = 100$



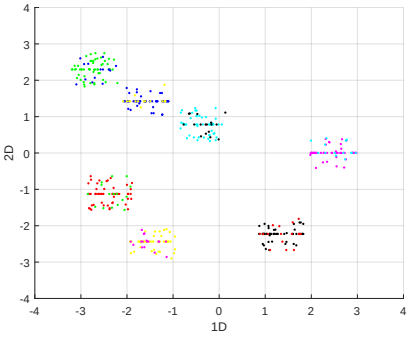
(b) $LearningStep = 200$



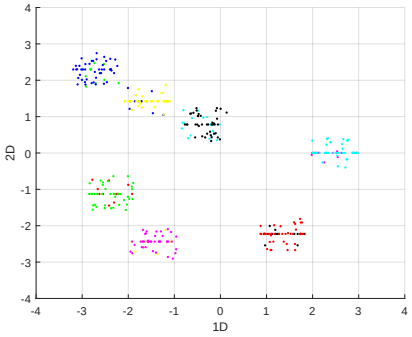
(c) $LearningStep = 300$



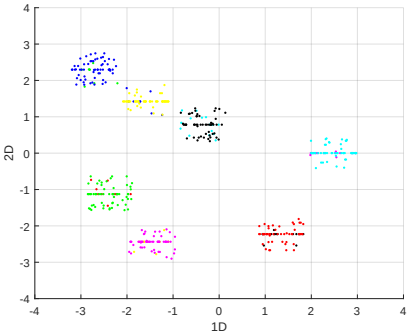
(d) $LearningStep = 500$



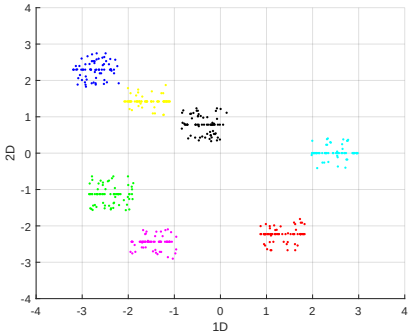
(e) $LearningStep = 600$



(f) $LearningStep = 800$

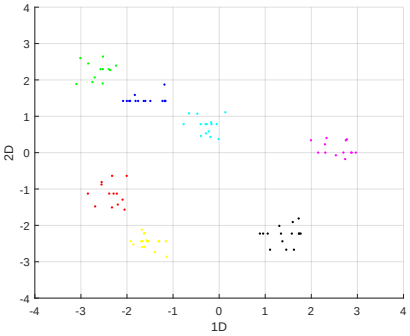


(g) $LearningStep = 900$

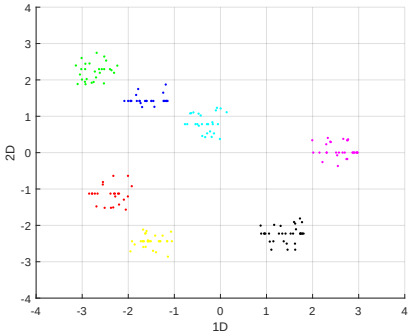


(h) $LearningStep = 980$

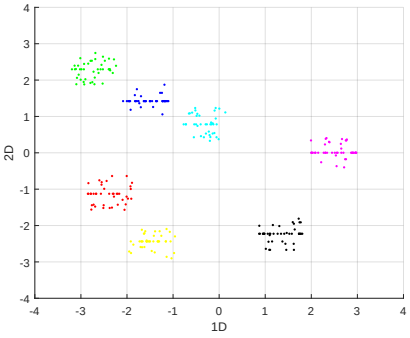
Figure 6.43: CALM-nepLRB EKB Transition on Data 6 ($L = 5$)



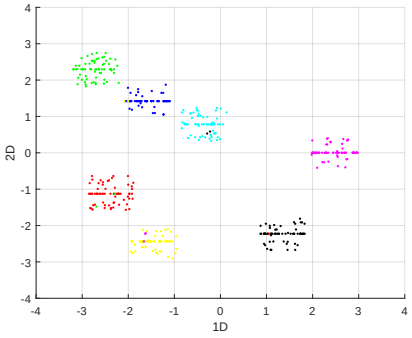
(a) $LearningStep = 100$



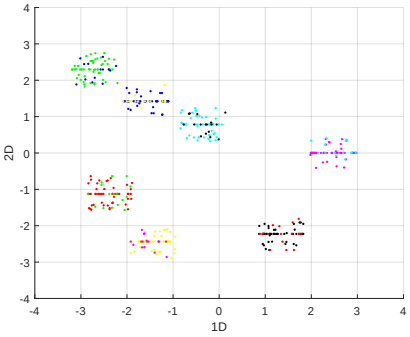
(b) $LearningStep = 200$



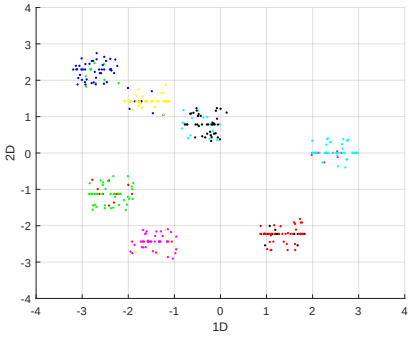
(c) $LearningStep = 300$



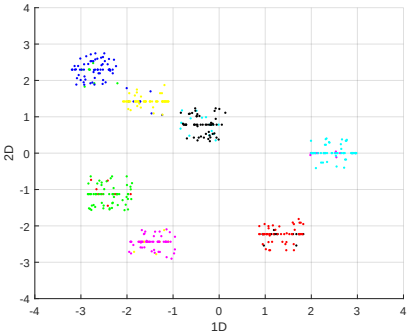
(d) $LearningStep = 500$



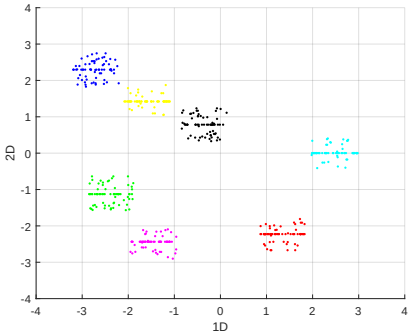
(e) $LearningStep = 600$



(f) $LearningStep = 800$



(g) $LearningStep = 900$



(h) $LearningStep = 980$

Figure 6.44: CALM-nepLRB EKB Transition on Data 6 ($L = 6$)

By looking into the EKB status transition of each algorithm with each CALM-ANN, we can understand more clearly how the incremental accuracy comes out from the previous experiments. Figure 6.30, Figure 6.31, Figure 6.32, Figure 6.33, and Figure 6.34 show EKB status transition of CALM-eLRB with each CALM-ANN when the learning steps are 100, 200, 300, 500, 600, 800, 900, and 980. Figure 6.35, Figure 6.36, Figure 6.37, Figure 6.38, and Figure 6.39 show EKB status transition of CALM-epLRB with each CALM-ANN when the learning steps are 100, 200, 300, 500, 600, 800, 900, and 980. Likewise, Figure 6.40, Figure 6.41, Figure 6.42, Figure 6.43, and Figure 6.44 show EKB status transition of CALM-nepLRB with each CALM-ANN when the learning steps are 100, 200, 300, 500, 600, 800, 900, and 980.

For CALM-eLRB, Figure 6.31d and Figure 6.32d show that CALM-ANN2 and CALM-ANN3 successfully stored its experiences on EKB at learning step 500 which is same as the unchanged data subset as shown Figure 6.27a. This supports the experimental results of the incremental dynamic accuracy. As shown on Figure 6.28c and Figure 6.29c, CALM-eLRB with CALM-ANN2 and CALM-ANN3 reached 100% accuracy on unchanged data subset which is consistent in transition of the EKB.

Figure 6.33d shows that CALM-eLRB with CALM-ANN5 could not fully stored the unchanged data subset compared to it with CALM-ANN2 and CALM-ANN3. This also supports the experimental results that CALM-eLRB with CALM-ANN4 reached around 85% in incremental dynamic accuracy as shown Figure 6.28c.

Figure 6.30d and Figure 6.34d show very interesting points that CALM-eLRB with CALM-ANN1 and CALM-ANN5 accumulated mix-matched experiences of both unchanged and changed environment at the end of learning step, 980. First

of all, in Figure 6.30d, we can see CALM-eLRB with CALM-ANN1 could learn only partial experiences of unchanged data subset at learning step 500. This means CALM-ANN1 can not achieve high accuracy of unchanged data subset as shown Figure 6.28c. After learning step 500, CALM-ANN1 starts to learn newly changed environment and partially adapted changed environment as shown in Figure 6.30h. This supports that CALM-ANN1 did not dramatically drop its incremental dynamic accuracy as shown in Figure 6.28c.

Likewise, in Figure 6.34d, we can see CALM-eLRB with CALM-ANN5 learned only partial experiences of unchanged data subset at learning step 500 however it started to learn newly changed environment and accumulated the new experiences as shown in Figure 6.34h. And actually, CALM-eLRB with CALM-ANN5 stacked more experiences of changed environment than unchanged one. This supports the result that CALM-ANN5 shows increasing accuracy after learning step 500 as shown in Figure 6.28c.

For CALM-epLRB, Figure 6.36d, Figure 6.37d, and Figure 6.37d show that CALM-epLRB with CALM-ANN2, CALM-ANN3, and CALM-ANN4 could successfully learn the unchanged environment, which are exactly same as Figure 6.27a. This supports that CALM-eLRB with CALM-ANN2, CALM-ANN3, and CALM-ANN4 reached 100% incremental dynamic accuracy at learning step 500.

Figure 6.35d shows that CALM-ANN1 could partially understand the unchanged environment and also stacked new experiences from changed environment at the end of learning step. This also supports the incremental dynamic accuracy of CALM-ANN1 is steady in the Figure 6.28e.

Figure 6.39d shows that CALM-ANN5 could partially understand the unchanged environment but could less understand about changed environment

at the end of the learning step 980. When we compare the Figure 6.39c and Figure 6.39d, we can see CALM-ANN6 accumulated more successful experiences between learning step 300 and 500. This supports the dynamic accuracy of CALM-ANN6 dramatically increased between learning step 400 and 500 as shown in Figure 6.28e. Also after learning step 500 the dynamic accuracy is decreased and this is supported that CALM-ANN6 stacked more experiences from unchanged environment.

For CALM-nepLRB, Figure 6.40d, Figure 6.41d, Figure 6.42d, Figure 6.43d, and Figure 6.44d show that all CALM-ANNs could successfully learn the unchanged environment which are mostly same as Figure 6.27a. This supports the experimental results of the incremental dynamic accuracy that all of CALM-ANN shows high accuracy at learning step 500 as shown in Figure 6.28g. It is notable that all CALM-ANNs of CALM-nepLRB started to learn after the environment is changed by changing old experience with new ones and eventually they could successfully learn the changed environment by memorizing the changed experiences as shown in the Figure 6.40h, Figure 6.41h, Figure 6.42h, Figure 6.43h, and Figure 6.44h. This supports the Figure 6.29h which shows most of CALM-ANNs increased its accuracy on changed data subset.

Chapter 7

Discussions

This chapter provides six discussion topics based on the CALM experiments and results. Especially, this chapter discusses how the different learning results of each algorithm are related to some specific CALM characteristics.

7.1 Issue 1: The Meaning of Accumulated Rewards

First issue is the relationship between accumulated rewards and accuracy. We note that increasing accumulated rewards does not guarantee the incremental increasing accuracy. This fact reminds us of two important features of learning process: (1) getting reward at current learning step is only regarding to current context; (2) however, evaluating accuracy at each learning step considers all covered training data up until current step or whole testing data. This means even if the accumulated reward are increasing, this is not guarantee to successfully recognize given all contexts at the end of the learning steps. For example, in Figure 6.21, all of the algorithms with all different neural networks show the monotonically increasing graph; however all the accuracy at the end of the learning steps is mostly around 20% as shown in Figure 6.10. Even more, CALM-nepLRB shows decreasing accuracy while the accumulated reward are growing as shown in Figure 6.10h. Therefore, in analyzing accumulated rewards, what we can get from the information is that how many times an algorithm could

make good choices through all of the learning steps; but we are not supposed to overlook the possibility of gradually increasing accumulated rewards with low accuracy performance.

7.2 Issue 2: The Role of Depth

Second issue is the relationship between the depth of a neural network and accuracy. We conclude that deepest neural network does not guarantee the highest performance. Superficially, it can be easily considered that having deeper layer causes higher performance. However, we should not overlook the fact that having more layers makes larger search space with more weight vectors. In this regard, gradient-descent optimization might be stuck in a local optimum, which is not guarantee to find the global optimum. For example, from Figure 6.2 to Figure 6.11, it is not black line representing 6-layered neural network that shows highest accuracy in the experiments of CALM-rLRB, CALM-eLRB, and CALM-epLRB. CALM-nepLRB is an exception since it has more advanced features compared to the others. In CALM-nepLRB, black line is also mostly survived in making high performance as well as the other colors. Therefore, except for CALM-nepLRB, which is a novel bio-inspired generalized arbitrary-depth neural controller with additional neurobiological features, setting deeper neural network does not always come up with high performance in the non-bio-inspired algorithms.

7.3 Issue 3: The Magic Number 3

Third issue is about magic number three and ambiguous number two. There are two notable phenomenon from the experiments in Chapter 6: (1) 2-layered

neural network has limitations to represent the given input spaces with a certain number of iterations and (2) 3-layered neural network seems to be generally enough to solve the given input data sets. For example, from Figure 6.2 to Figure 6.11, we can see yellow lines representing 3-layered neural networks are mostly above of the other colored lines through all of the experiments regardless of number of iterations. On the other hand, there are two ways of interpreting the role of 2-layered neural network depending on the number of iterations. First of all, if the number of iteration is 1, the blue lines representing 2-layered neural networks stays relatively high accuracy compared to the other lines. However, if it is 200, the blue lines are stays usually lowest level of accuracy than the others. We suggest three conclusions from these phenomenon: (1) in low frequency optimization, simplest neural network tends to be less confused in finding right answers based on experiences, (2) in high frequency optimization with large number of iteration, the simplest neural network is limited to find all right answers corresponding to the given input, and (3) usually 3-layered neural network with 25 hidden nodes are enough to make high accuracy on the generated synthetic data sets in this dissertation described in Chapter 6.

7.4 Issue 4: CALM-eLRB vs CALM-epLRB

Fourth issue is about the role of Selective-Power-Update. We can see the performances from CALM-eLRB and CALM-epLRB are comparable. It is actually hard to tell which one is better than the other one. For example, on DATA4 with Figure 6.8d, Figure 6.8f, Figure 6.9d, and Figure 6.9f, we can see magenta lines representing 4-layered networks and black lines show more consistent accuracy in CALM-epLRB but yellow lines show higher accuracy in CALM-eLRB. Similarly, on DATA3 with Figure 6.6d, Figure 6.6f, Figure 6.7d,

and Figure 6.7f, we can see cyan lines representing 5-layered neural networks and black lines show better results in CALM-eLRB than CALM-epLRB but yellow and magenta lines show higher accuracy in CALM-epLRB than CALM-eLRB. In this regard, we tentatively conclude that the efficacy of the Selective-Power-Update gives an advantage when CALM-epLRB finds the similar context from EKB which successfully powers current network with positive reward. In other words, if CALM-epLRB finds the most similar context from EKB but it actually has different desired output, it will selectively power anti-desired weight vector since the similarity is calculated from only Euclidean distance between two data points. This phenomenon happened when two clusters are overlapped where they have each different desired output. For example, in the middle of the Figure 6.1c, we can see black cluster and magenta cluster is overlapped and the data points withing overlapped area are considered as similar contexts but actually having different desired output respectively.

7.5 Issue 5: When to Use CALM-rLRB

Fifth issue is about the usage of CALM-rLRB. As we have seen in Chapter 6 CALM-rLRB is not practical algorithm in a long term since it just aims to overcome current context. However, it is beneficial when the experience-based optimization can not recognize current context due to the role of back neurons in CALM-nepLRB. For example, if currently selected output is $behavior3 - GoForward$ and it gets zero reward, the dopamine will be released and the back neuron $behavior4 - GoBackward$, which is opposite action of $behavior3$, then theoretically the same context will be come up again. In this case, it is possible for CALM-nepLRB to repeat selecting $behavior3 - GoForward$ infinitely since the optimization process for CALM-eLRB or CALM-epLRB keep are based on

the same saved past positive experiences. In other words, if the same context comes again due to the effect of back neurons, CALM-nepLRB uses the same learning input and quasi-target output for the optimization and thus it will cause same behavior which caused zero reward. In this case, CALM-rLRB is the essential solution because it ignores the past positive experiences but focuses only on current information in the optimization process. This is the reason why CALM-nepLRB has an ability to select an appropriate algorithm based on learning status. In this way, CALM-rLRB is useful when an algorithm is supposed to ignore the experiences in a short term. Biologically, it can be also said that CALM-rLRB is an appropriate algorithm for short-term memory while the others are profitable for long-term memory in a vertebrate brain. This should be discussed based on more study on memory-related future research works.

Chapter 8

Conclusions

8.1 Conclusions

This research introduces CALM (Context-Aware Learning Model) including four different learning algorithms, which is a novel context-aware learning model inspired by (1) supervised learning with logistic regression backpropagation, (2) modulatory hyperbolic reward-based learning, and (3) behavioral neurobiology with OBIBIDEEV features. This research describes detail features and background knowledge with sound mathematical derivation of each algorithm. The logistic regression and reward-based learning algorithms are addressed in depth with the generalized format. The essential study of the basic neurobiology and behavioral neural circuits are investigated to provide general validity of building a robot brain CALM-nepLRB with CALM-nepLRB-ANN: bat, owl, eel, crayfish, honeybee, drosophila, and moth. CALM is evaluated with five types of measurement on six synthetic data sets, which shows that CALM-eLRB, CALM-epLRB, and CALM-nepLRB are promising; and, it is demonstrated that CALM-nepLRB outperforms all of these other algorithms.

8.2 Contributions

The research contributions in this dissertation can be roughly summarized as five parts. First contribution of this research is in building a novel context-aware learning model with OBIBIDEEV learning features: (1) Online, (2) Bootstrapping, (3) Interactive, (4) Bio-inspired, (5) Incremental, (6) Dynamic, (7) Experience-based, (8) Experience-powered, (9) Arbitrary depth which can be applied in non-robotics, robotics, and neurorobotics areas. Second contribution is in providing neurobiological backgrounds supporting the robot brain, which are not usually covered by the preceding research works. Third contribution is providing solid mathematical derivation and deep understanding for research background. Fourth contribution is providing appropriate pseudocodes algorithms for each learning type in a generalized format. Fifth contribution is to evaluate the learning model in a generalized way with several synthetic data sets so that we can have a chance to compare and discuss its performance simply and clearly before applying it into a real world.

- CALM serves as a context-aware learning middleware which can be applied into a general learning areas. The system architecture are designed in details and the role of each component is clear so that a learner can utilize it with a little efforts in building one's own learning model. Providing a well-designed context-aware neural middleware is a contribution.

- CALM is unique framework in that it has all OBIBIDEEV features based on reward-based neuromodulatory optimization with promising performance which is demonstrated on the six synthetic data sets. In other words, CALM is a novel hybrid learning model which makes promising performance by

overcoming the limitations on existing typical machine learning approach. It is not about just taking several existing features to make a new learning model, but it is about building a new learning type which is based on profound understanding of existing approaches and newly investigated neurobiological features in neurorobotics area. Successfully adhering to the supervised optimization approach based on reward and experience with a generalized model is a significant contribution.

- CALM is a generalized brain. This research introduces a novel, bio-inspired, generalized, arbitrary-depth, neural network, CALM-nepLRB-ANN, and provides a distinctive neuromodulatory algorithm, CALM-nepLRB, which has flexibility to select appropriate algorithm based on reward and experiences. This research demonstrates the power of a brain with additional neurobiological features: combination-sensitive neurons, recurrent inhibition, two types of dopaminergic neuromodulation. Successfully incorporating the features into learning model which eventually causing highly promising performance in a generalized way is also a significant contribution.

- CALM is based on completely generalized mathematical descriptions with clear derivations. Especially, this research provides profound understanding of the logistic regression and reward-based learning with each corresponding clear pseudocode algorithm in a unique way. Describing artificial neural networks with comparing a general linear algebra and scientific computations is also fine contribution.

- CALM is evaluated from artificially generated synthetic data sets which covers different combination of possible input problems. This provides a chance to easily test and compare its performances on various domains in a generalized way before applying it into real world.

Chapter 9

Future Work

This chapter provides the future works for this research in two perspectives.

9.1 Increasing Feasibility

CALM should be evaluated in a virtual environment with a robot simulation such as a TurtleBot in Gazebo using ROS (Robot Operating System). In this dissertation, CALM is only tested with synthetic data sets to show its performance; therefore, its promising performance and practical usages should be also demonstrated by applying it into either virtual simulating environment or a real environment with a real robot.

9.2 Increasing Reliability

CALM should be also tested with more input features with various types of input in order to verify its ability of multi-modal handling. In this dissertation, only two input features plus a bias node are used and only one type of input, Cartesian coordinate information, is used, which is generated artificially. Along with this, several types of input should be processed as contextual information from Context Supplier in a more sophisticated way in order to demonstrate high-quality neural context-awareness. Currently, Context Supplier directly takes the input from Sensory System as a context, which should be advanced in

future works.

CALM should be evaluated with more values of learning parameters. For example, in this dissertation, there are fixed value of learning parameters: $\eta = 1.0,\ \lambda = 0.0,\ \epsilon = 0.3,\ \gamma = 0.5$, and $ITR \in \{1, 200\}$. Further experiments on same data but with different values of the parameters can give us best set of learning parameters which cause best performance of CALM.

Study of neurobiological on hippocampus of a vertebrate brain should be accomplished to compare the role of EKB (Experience-based Knowledge Base) with the functions of the hippocampus; this further study can support a robot brain not only computationally but also neurobiologically. In this dissertation, EKB has two important roles: (1) retrieving relevant experience-based on current contextual information with Euclidean distance sequential search engine and (2) optimizing knowledge base by finding most recent experience instead of old experience when both of them are mostly similar to the current contextual information. Also, in the future work, if the search engine is based on more advanced algorithm with lower computational complexity and if more advanced knowledge base optimization approaches are studied, the as above CALM would be increased with high efficiency of using the past positive experiences in CALM.

9.3 Performance Analysis

It is clear that more sophisticated algorithms take more time to learn an environment. For example, CALM-nepLRB took more time to get the experimental results compared to the other the other algorithms; CALM-eLRB and CALM-epLRB had more running times than CALM-rLRB; CALM-rLRB had very short time compared to the other algorithms since it did not use experiences for its optimization. Therefore, it will be worth of investigating algorithm

running time of each CALM algorithm for each experiment thus we can compare each algorithm performance compared to running time in the future.

Bibliography

[1] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer-Verlag, 1999.

[2] Dacks AM, Christensen TA, and Hildebrand JG. Modulation of olfactory information processing in the antennal lobe of manduca sexta by serotonin. *Neurophysiology*, 99:2077–2085, 2008.

[3] Anil K. Seth and Olaf Sporns and Jeffrey L. Krichmar. Neurorobotic models in neuroscience and neuroinformatics. *Neuroinformatics*, 3(3):167–170, September 2005.

[4] Kyle T. Beggs and Alison R. Mercer. Dopamine receptor activation by honey bee queen pheromone. *Current Biology*, 19(14):1206–1209, 2009.

[5] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009.

[6] Bruce A. Carlson and Masashi Kawasaki. Stimulus selectivity is enhanced by voltage-dependent conductances in combination-sensitive neurons. *Neurophysiology*, 96(6):3362–3377, December 2006.

[7] M. C.Avery, D.A. Nitz, A.A.Chiba, and J. L. Krichmar. Simulation of cholinergic and noradrenergic modulation of behavior in uncertain environments. *Frontiers in Computational Neuroscience*, 6:1–16, 2012.

[8] Brian R. Cox and Jeffrey L. Krichmar. Neuromodulation as a robot controller: a brain inspired strategy for controlling autonomous robots. *IEEE Robotics and Automation Magazine*, 16:72–80, 2009.

[9] A. M. Dacks, J. A. Riffell, J. P. Martin, S. L. Gage, and A. J. Nighorn. Olfactory modulation by dopamine in the context of aversive learning. *Neurophysiology*, 108(2):539–550, July 2012.

[10] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human Computer Interaction (HCI)*, pages 97–166, 2001.

[11] Andries P. Engelbrecht. *Computational Intelligence.* Wiley, second edition, 2007.

[12] I. H. Suh et al. Ontology-based multi-layered robot knowledge framework (omrkf) for robot intelligence. pages 429—-436. IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS), 2007.

[13] Purves D. et al. *Neuroscience*. Sinauer, fifth edition, 2012.

[14] Jason G. Fleischer and Gerald M. Edelman. Brain-based devices: An embodied approach to linking nervous system structure and function to behavior. *IEEE Robotics and Automation Magazine*, 6(3):33–41, September 2009.

[15] Dario Floreano and Claudio Mattiussi. *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. The MIT Press, 2008.

[16] Donald Olding Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Wiley, New York, 1949.

[17] Wonil Hwang, Jinyoung Park, Hyowon Suh, Hyungwook Kim, and IlHong Suh. Ontology-based framework of robot context modeling and reasoning for object recognition. pages 596–606. Springer Berlin Heidelberg, 2006.

[18] Bergan JF, Ro P, Ro D, and Knudsen EI. Hunting increases adaptive auditory map plasticity in adult barn owls. *Neuroscience*, 25(42):9816–9820, October 2005.

[19] I.T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

[20] Eric I. Knudsen. Instructed learning in the auditory localization pathway of the barn owl. *Nature*, 417(6886):322–328, May 2002.

[21] J. L. Krichmar. A biologically inspired action selection algorithm based on principles of neuromodulation. pages 1–8. International Joint Conference on Neural Networks (IJCNN), 2012.

[22] J. L. Krichmar. A neurorobotic platform to test the influence of neuromodulatory signaling on anxious and curious behavior. *Frontiers in Neurorobotics*, 7(1):1–17, 2013.

[23] Jeffrey Krichmar and Florian Röhrbein. Value and reward based learning in neurorobots. *Frontiers in Neurorobotics*, 7:13, 2013.

[24] Jeffrey L. Krichmar. The neuromodulatory system: A framework for survival and adaptive behavior in a challenging world. *Adaptive Behavior*, 16(6):385–399, December 2008.

[25] Irving Kupfermann. Modulatory actions of neurotransmitters. *Annual Review of Neuroscience*, 2:447–465, 1979.

[26] G. Lim and I. Suh. Weighted action-coupled semantic network (wasn) for robot intelligence. pages 2035—-2040. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008.

[27] G. H. Lim, I. H. Suh, and H. Suh. Ontology-based unified robot knowledge for service robots in indoor environments. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 41(3):492–509, May 2011.

[28] Gi Hyun Lim, Il Hong Suh, and Luís Seabra Lopes. The representation of weighted action-coupled semantic network and spreading activation model for improvisational action. In *SMC*, pages 4054–4059. IEEE, 2013.

[29] Brie A. Linkenhoker and Eric I. Knudsen. Incremental training increases the plasticity of the auditory space map in adult barn owls. *Nature*, 419:293–296, September 2002.

[30] George F. Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Addison-Wesley, fifth edition, 2004.

[31] Schwaerzel M, Monastirioti M, Scholz H, Friggi-Grelin F, Birman S, and Heisenberg M. Dopamine and octopamine differentiate between aversive and appetitive olfactory memories in drosophila. *Neuroscience*, 23:10495–10502, 2003.

[32] Philip Moore and Hai V. Pham. Predicting intelligence using hybrid artificial neural networks in context-aware tunneling systems under risk and uncertain geological environment. pages 989–994. Complex, Intelligent, and Software Intensive Systems (CISIS), July 2012.

[33] S. Nakanishi, T. Hikida, and S. Yawata. Distinct dopaminergic control of the direct and indirect pathways in reward-based and avoidance learning behaviors. *Neuroscience*, 282:49–59, 2014.

[34] Kuniaki Noda, Hiroaki Arie, Yki Suga, and Tetsuya Ogata. Multimodal integration learning of robot behavior using deep neural networks. *Robotics and Autonomous Systems*, 62:721–736, 2014.

[35] Kloppenburg P and Mercer AR. Serotonin modulation of moth central olfactory neurons. *Annu Rev Entomol*, 53:179–190, 2008.

[36] Ivan Petrovich Pavlov. *Lectures on Conditioned Reflexes*. Liveright, New York, 1928. Translated by W. Horsley Gantt.

[37] Jose L. Pena and Yoram Gutfreund. New perspectives on the owl's map of auditory space. *Current opinion in neurobiology*, pages 55–62, 2014.

[38] C.M.A. Pennartz. Reinforcement learning by Hebbian synapses with adaptive threshold. *Neuroscience*, 81(2):303–319, 1997.

[39] Akimul Prince and Biswanath Samanta. Neuromodulation based control of an autonomous robot. In *International Joint Conference on Neural Networks*, pages 1–7. IEEE, 2013.

[40] Menzel R, Heyne A, Kinzel C, Gerber B, and Fiala A. Pharmacological dissociation between the reinforcing, sensitizing, and response-releasing functions of reward in honeybee classical conditioning. *Behavioral Neuroscience*, 113(4):744–754, August 1999.

[41] Alan Roberts. Recurrent inhibition in the giant-fibre system of the crayfish and its effect on the excitability of the escape response. *Experimental Biology*, 48:545–567, 1968.

[42] Sebastian Rockel, Bernd Neumann, Jianwei Zhang, Krishna Sandeep Reddy Dubba, Anthony G. Cohn, Stefan Konecny, Masoumeh Mansouri, Federico Pecora, Alessandro Saffiotti, Martin Günther, Sebastian Stock, Joachim Hertzberg, Ana Maria Tomé, Armando J. Pinho, Luís Seabra Lopes, Stephanie von Riegen, and Lothar Hotz. An ontology-based multi-level robot architecture for learning from experiences. In *Designing Intelligent Robots: Reintegrating AI II, Papers from the 2013 AAAI Spring Symposium, Palo Alto, California, USA, March 25-27, 2013*. AAAI, 2013.

[43] Stuart Russell and Peter Norvig. *Artificial Intelligence a Modern approach*. Prentice Hall, second edition, 2002.

[44] Sareh Saeedi, Tom Carlson, Ricardo Chavarriaga, and Jose del R. Millan. Making the most of context-awareness in brain-computer interfacesd. pages 68–73. IEEE International Conference on Cybernetics, 2013.

[45] Lisa M. Saksida, Scott M. Raymond, and David S. Touretzky. Shaping robot behavior using principles from instrumental conditioning. *Robotics and Autonomous Systems*, 22(3-4):231–249, December 1997.

[46] R. Salgado, F. Bellas, P. Caamaño, B. Santos-Díez, and R. J. Duro. A procedural long term memory for cognitive robotics. In *Evolving and Adaptive Intelligent Systems (EAIS), 2012 IEEE Conference on*, pages 57–62. Evolving and Adaptive Intelligent Systems (EAIS), May 2012.

[47] JT Sanchez, D Gans, and JJ Wenstrup. Glycinergic "inhibition" mediates selective excitatory responses to combinations of sounds. *Neuroscience*, 28(1):80–90, January 2008.

[48] Bill Schilit and Marvin Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8:22–32, 1994.

[49] Bill N. Schilit, Norman Adams, Rich Gold, Michael M. Tso, and Roy Want. The parctab mobile computing system. In *Workshop on Workstation Operating Systems*, pages 34–39, 1993.

[50] Ashvin Shah. *Psychological and Neuroscientific Connections with Reinforcement Learning*, pages 507–537. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[51] M.W. Shiflett and B.W. Balleine. Contributions of erk signaling in the striatum to instrumental learning and performance. *Behavioral Brain Research*, 218(1):240–247, 2011.

[52] L. N. Soldatova, A. Clare, A. Sparkes, and R. D. King. An ontology for a robot scientist. *Bioinformatics*, 22(14):e464–e471, July 2006.

[53] Andrea Soltoggio and Kenneth O. Stanley. From modulated Hebbian plasticity to simple behavior learning through noise and weight saturation. *Neural Networks*, 34:28–41, October 2012.

[54] Olaf Sporns and William H. Alexander. Dopamine, reward conditioning, and robot behavior. In *Proceedings of the 2Nd International Conference on Development and Learning*, ICDL '02, pages 265–, Washington, DC, USA, 2002. IEEE Computer Society.

[55] Olaf Sporns and William H. Alexander. Neuromodulation and plasticity in an autonomous robot. *Neural Network*, 15(4):761–774, June 2002.

[56] Joohee Suh and Dean F. Hougen. Context-based adaptive robot behavior learning model (carb-lm). pages 206–211. IEEE Symposium Series on Computational Intelligence, December 2014.

[57] RS. Sutton and AG. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, USA, 1998.

[58] Rick P. Thomas, Michael R. Dougherty, Amber M. Sprenger, and Harbison J. Isaiah. Diagnostic hypothesis generation and human judgment. *Psychological Review*, 115(1):155–185, January 2008.

[59] Eiji Uchibe and Kenji Doya. *Finding Exploratory Rewards by Embodied Evolution and Constrained Reinforcement Learning in the Cyber Rodents*, pages 167–176. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[60] Nachum Ulanovsky and Cynthia F. Moss. What the bat's voice tells the bat's brain. pages 8491—8498. Proceedings of the National Academy of Sciences of the United States of America, 2008.

[61] Florian R. V. A reinforcement learning algorithm for spiking neural networks. pages 299–306. Proceedings of the Seventh International Symposiumon Symbolic and Numeric Algorithms for Scientific Computing, 2005.

[62] M.C.W. van Rossum, G.Q. Bi, and G.G. Turrigiano. Stable Hebbian learning from spike timing-dependent plasticity. *Neuroscience*, 20(23):8812–8821, 2000.

[63] Scott Waddell. Dopamine reveals neural circuit mechanisms of fly memory. *Trends in Neurosciences*, 33(10):457–464, October 2010.

[64] C.J.C.H. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge, England, 1989.

[65] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, September 1991.

[66] Mark Weiser. The computer for the 21st century. *ACM SIGMOBILE Mobile Computing and Communications Review*, 3(3):3–11, July 1999.

[67] Shin-Rung Yeh, Russell A. Fricke, and Donald H. Edwards. The effect of social experience on serotonergic modulation of the escape circuit of crayfish. *Science*, 271(5247):366, January 1996.

[68] G. Zupanc and T. Bullock. Walter heiligenberg: the jamming avoidance response and beyond. *Comparative Physiology A: Neuroethology, Sensory, Neural, and Behavioral Physiology*, 192(6):561–572, June 2006.