GENEREC IMPLEMENTATION OF THE CORTICO-HIPPOCAMPAL MODEL OF GLUCK AND MYERS IN HIPPOCAMPAL REGION ATROPHY

by

İlim Çağıran

B.S. in Chemical Engineering, Boğaziçi University, 2002

Submitted to the Institute of Biomedical Engineering
in partial fulfillment of the requirements
for the degree of
Master of Science
in
Biomedical Engineering

Boğaziçi University

February, 2006

GENEREC IMPLEMENTATION OF THE CORTICO-HIPPOCAMPAL MODEL OF GLUCK AND MYERS IN HIPPOCAMPAL REGION ATROPHY

by

İlim Çağıran

APPROVED BY:

Assoc. Prof. Dr. Halil Özcan Gülçür ……
(Thesis Supervisor)

Assoc. Prof. Dr. Hale Saybaşılı ……
(Thesis Co-Supervisor)

Assoc. Prof. Dr. Yasemin P. Kahya ……

Prof. Dr. Tamer Demiralp ……

Asst. Prof. Dr. Burak Güçlü ……

DATE OF APPROVAL: June 8, 2006

ACKNOWLEDGMENTS

I am indebted to many people who helped complete this work.

In particular, I am most grateful to my thesis supervisor Assoc. Prof. Dr. H. Özcan Gülçür for his great effort to support my progression on this work in my hard times, his help on every detail of this thesis and his invaluable encouragement.

Special thanks to my thesis co-supervisor Assoc. Prof. Dr. Hale Saybaşılı with her support and tolerance in all phases of this work. This work could not be realized without her advices, encouragements and her classes that provided the foundations of this work.

I am grateful to Assoc. Prof. Dr. Randall C. O'Reilly, University of Colorado Boulder, Institute of Cognitive Science Center for Neuroscience, for the counseling he provided via internet during the implementation of the model. I would also like to thank to Asst. Prof. Dr. Catherine E. Myers, Rutgers University-Newark, for her help on providing selected papers which would not be available otherwise. I wish to thank Asst. Prof. Dr. Burak Güçlü for his helpful suggestions on my thesis.

I would like to acknowledge my sincere thanks to my family for their support, not just throughout this study but also at each stage of my life. I would wish my father was still alive and could proud of this work, since he is the reason of being the one I am now.

Last but not least, thanks to all of my close friends for their tolerance and encouragement.

GENEREC IMPLEMENTATION OF THE CORTICO-HIPPOCAMPAL MODEL OF GLUCK AND MYERS IN HIPPOCAMPAL REGION ATROPHY

ABSTRACT

Hippocampal region dysfunction is suggested to have an important effect for the cognitive impairments observed in Alzheimer's disease. In some patients, hippocampus and nearby structures show atrophy while other brain structures appear intact. Hence, study of neural network models which can mimic biological and psychological findings is hoped to contribute to our understanding of the underlying reasons and possible consequences of hippocampal dysfunction. Therefore the main objective of this thesis work was to develop an artificial neural network model that in many ways behaved like the hippocampal region. For this purpose we have used the cortico-hippocampal model of Gluck and Myers as the basic model. The learning rule Gluck and Myers used in their original work was backpropagation. Hoping to get a more biologically plausible model, the learning rule was changed to generalized recirculation (GeneRec). Furthermore, instead of using negative weights, the network was externally inhibited by two alternate methods: the kWTA inhibition and via additional inhibitory interneurons. Also, a weight bounding function was applied to the weight update rules.

Addition of external inhibition and weight bounding functions to the network reduced the convergence characteristics of the network. Particularly cortico-cerebellar side of the network could not converge with external inhibition. Therefore external inhibition was abandoned for the cortico-cerebellar side. Although the hippocampal network could converge with kWTA, inhibition and weight bounding, rapid changes of activations of hippocampal network hidden layer neurons during training caused huge oscillations on the cortico-cerebellar output. Hence, external inhibition was abandoned also for the hippocampal network.

The results of several representational differentiation and compression cases were found comparable to the Gluck and Myers original work.

Keywords: Hippocampus, model, hippocampal atrophy, neural network, generalized recirculation

HİPOKAMPÜS BÖLGESİ ATROFİSİ İÇİN GLUCK VE MYERS'İN KORTEKS-HİPOKAMPÜS MODELİNİN GENEREC UYARLAMASI

ÖZET

Hipokampüs bölgesindeki işlev bozuklukları Alzheimer hastalığında gözlenen bilişsel bozukluklarda önemli bir etken olarak görülmektedir. Bazı hastalarda salt hipokampüs ve yakınındaki yapılarda atrofi gözlemlenirken diğer beyin yapıları bozulmadan kalabilmektedir. Biyolojik ve fizyolojik bulguların sonuçlarını taklit eden sinir ağı modelleri ile yapılan çalışmaların, hipokampüs işlev bozukluğunun altında yatan nedenleri ve olası sonuçlarını anlamaya katkı yapacağı umulmaktadır. Bu nedenle bu tez çalışmasının ana hedefi, birçok yönden hipokampüs bölgesi gibi davranan yapay bir sinir ağı modeli geliştirmektir. Bu amaçla Gluck ve Myers'ın korteks-hipokampüs sinir ağı modeli temel model olarak seçilmiştir. Gluck ve Myers çalışmalarında öğrenme kuralı olarak geri-yayılım kuralını kullanmışlardır. Biyolojik kabul edilebilirliği daha yüksek bir model elde etmek ümidiyle öğrenme kuralı Genelleştirilmiş Yeniden Dolaşım (GeneRec) kuralı ile değiştirilmiştir. Ayrıca, eksi ağırlıklar kullanmak yerine sinir ağı kWTA engellemesi ve engelleyici ara-nöronlar olarak seçenek iki yöntem ile dışarıdan engelleme yapılmıştır. Ağırlık yenileme kuralına bir de ağırlık sınırlama işlevi uygulanmıştır.

Ağa dış engelleme ve ağırlık sınırlama işlevlerinin eklenmesi ağın yakınsama özelliğini azaltmıştır. Özellikle ağın korteks-serebrum kısmı dış engelleme ile yakınsanamamıştır. Bu yüzden korteks-serebrum kısmında dış engelleme kullanımı terk edilmiştir. Hipokampüs bölgesine karşılık gelen ağ kWTA engellemesi ve ağırlık sınırlaması ile yakınsanabilse de hipokampüs ağının eğitim sırasında gizli katmanlarındaki sinir hücrelerinin etkinlikleri hızla değiştiği için korteks-serebrum çıktısında çok büyük salınımlara neden olmuştur. Bu nedenle hipokampüs ağında da dış engelleme kullanımı terk edilmiştir.

Çeşitli betimlemede farklılaşma ve sıkıştırma durumlarının sonuçları Gluck ve Myers'ın özgün çalışmalarıyla kıyaslanabilir sonuçlar vermiştir.

Anahtar Sözcükler: Hipokampüs, model, hipokampüs atrofisi, sinir ağı, genelleştirilmiş yeniden dolaşım kuralı

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Definition
CCN	Cortico/Cerebellar Network
HPN	Hippocampal Network
HR	Hippocampal Region
kWTA	k-Winners-Take-All
LTD	Long-Term Depretiation
LTM	Long-Term Memory
LTP	Long-Term Potentiation
SWB	Soft Weight Bounding

LIST OF SYMBOLS

Symbol	Name/Definition
ε	learning rate parameter
Δ_{ik}	weight change computed by the error driven algorithm
Δw	small change applied to w
$\Delta\beta$	small change applied to β
β_j	bias applied to neuron j
J	plus phase activations of hidden layer of cortical network neurons
x_i	sending unit activation of neuron i
y_j	receiving unit activation of neuron j
x^-, y^-	unit activation of neuron in the minus phase
x^+, y^+	unit activation of neuron in the plus phase
w_{ij}	synaptic weight of synapse i belonging to neuron j
$[x]_-$	operator returns x if $x<0$ and 0 otherwise
$[x]_+$	operator returns x if $x>0$ and 0 otherwise

1. INTRODUCTION

1.1 Motivation

Neuroscience covers a broad range of topics such as molecular and cellular studies and also human psychology and psychophysics. Computational modeling and theoretical analysis are essential tools to characterize nervous system, to determine how it functions and to understand how it operates under certain conditions. Descriptive, mechanistic, and interpretive models are needed to answer the questions: what, how, and why for the nervous system; to summarize vast amounts of experimental data compactly yet accurately and thus characterize behavior of neurons by themselves and as a system. These models may be founded loosely upon biophysical, anatomical, and physiological findings; however their main purpose is to describe phenomena, not to explain them. Mechanistic models concentrate on how the nervous systems operate on the basis of known anatomy, physiology and circuitry and therefore they can be used to form the bridge between descriptive models at various levels. Computational and information-theoretic principles are used to explore the behavioral and cognitive significance of a wide range of aspects of nervous system function by interpretive models to answer the question of why nervous systems operate as they do [1-9].

Most of the time, it is not easy to decide the appropriate level of modeling of a particular problem; a common misassumption is more detailed models are better. However, they must be detailed enough to make contact with the lower level yet simple enough to provide clear results at higher level.

1.2 Objectives

Hippocampal-region dysfunction has long been suggested to be an important contributor to the cognitive impairments observed in Alzheimer's disease (AD). Currently, it is a leading cause of death among people over the age of 60. Recent research has produced findings that may allow early detection of which individuals are most at risk to develop AD in the future. In some elderly individuals, the hippocampus and the endorhinal cortex show signs of atrophy while other nearby brain structures appear intact [1]. Hence, study of biologically plausible learning models will probably

contribute to understand the underlying reasons and possible consequences of hippocampal dysfunction related to these models. Therefore, the main objective of this thesis work is to develop an artificial neural network model that in many ways behaves like the hippocampal region. For this purpose we have used the cortico-hippocampal model of Gluck and Myers as the basic model. However, we modified this model by the application of the GeneRec algorithm with soft weight bounding and with external inhibition. This model was analyzed through extensive numerical simulations to study phenomena that occur in the hippocampal region, as this region undergoes atrophy.

1.3 Organization of the Thesis

Chapter 1 introduces the subject. A brief summary of the relevant background and a summary of the Cortico-Hippocampal Model of Gluck and Myers are presented in Chapter 2. Simulations and the results are given in Chapter 3. A detailed discussion of the simulation results are given in Chapter 4. Training Data Generator Codes, Cortico-Hippocampal Intact Model Codes and the Cortico-Hippocampal Lesion Model Codes are given in Appendix A, Appendix B and Appendix C, respectively.

2. THEORY

2.1 Introduction

A generally accepted phenomenon underlying learning and memory is activity dependent synaptic plasticity which plays a critical role in the development of neural circuits. For a complete understanding of functional and behavioral importance of synaptic plasticity, studies of how experience and training modify synapses, and how these modifications alters patterns of neuronal firing to affect behavior should be carried out. Continuing experimental studies may reveal ways in which neuronal activity can affect synaptic strength. Hence synaptic plasticity rules inspired from these studies have been applied to several tasks including auto- and heteroassociative memory, storage and recall of temporal sequences, pattern recognition, and function approximation.

In 1949, Donald Hebb conjectured that if input from neuron A often contributes to the firing of neuron B, then synapse from A to B should be strengthened. Hebb suggested that such synaptic modification could produce neuronal assemblies that reflect the relationships experienced during training. The Hebb rule forms the basis of much of the research done on the role of the synaptic plasticity in learning and memory.

Experimental work in a number of brain regions, including hippocampus, neocortex, and cerebellum, has revealed activity-dependent processes that can produce changes in the efficacies of synapses that persist for varying amounts of time. Changes in synaptic strength involve both transient and long-lasting effects. Changes that persist for more than one hour long require protein synthesis and called long-term potentiation (LTP). Another form of plasticity which is observed in cerebellum is long-term depression (LTD).

Studies of plasticity and learning involve analyzing how synapses are affected by activity over the course of a training period. There are three major types of learning procedures in unsupervised (also called self supervised) learning. A network responds to a series of inputs during training solely on the basis of its basic intrinsic connections and dynamics. The network than self organizes in a manner that depends on the synaptic

plasticity rule being applied and on the nature of the inputs presented during training [10].

In supervised learning, a desired set of input-output relationships is imposed on the network by a teacher during training. Networks that perform particular tasks can be constructed in this way by letting a modification rule adjust their synapses until the desired computation emerges as a consequence of the training process. This is an alternative to explicitly specifying the synaptic weights. In this case, finding a plausible teaching mechanism may not be concern if the question is being addressed is whether any weights can be found that allow a network to implement a particular function. In more biologically plausible examples of supervised learning, one network acts as the teacher for another network.

In reinforcement learning, that is intermediate of these cases, the network output is not constrained by a teacher, but evaluative feedback about network performance is provided in the form of reward or punishment [11]. This can be used to control the synaptic modification process.

Non-Hebbian forms of synaptic plasticity, such as those that modify synaptic strengths solely on the basis of pre- or postsynaptic firing, are likely to play important roles in the homeostatic, developmental, and learning processes. Activity can also modify the intrinsic excitability and response properties of neurons [12]. Models of such intrinsic plasticity show that neurons can be remarkably robust to external perturbations if they adjust their conductance to maintain specified functional characteristics. Intrinsic and synaptic plasticity can interact in interesting ways. For example, shifts in intrinsic excitability can compensate for changes in the level of input to a neuron caused by synaptic plasticity. It is likely that all of these forms of plasticity, and many others, are elements of both the stability and the adaptability of nervous system [3].

2.2 Cortico-Hippocampal Interaction in Associative Learning

Computational models of learning need to incorporate stimulus representations to allow appropriate generalization of learning between stimuli. The appropriate degree of generalization will depend on the particular problem, implying that representations

should be adaptable to suit current task demands. However, the computational resources required to create appropriate new stimulus representations on the fly are considerable; neural-network researchers have addressed this problem by developing the error back propagation algorithm.

However, it is not clear that the sophisticated neural machinery needed to create the necessary new stimulus representations exists throughout the brain. One possible evolutionary alternative would be to localize some of the mechanisms for representational change in a central location (such as cerebral cortex and hippocampus) so that other brain regions (such as cerebral cortex and cerebellum) could make use of these mechanisms as needed for particular tasks [13]. This idea forms the basis for two major models of hippocampal function.

In both of these models, one network module representing the hippocampal region interacts with other network modules representing other brain regions, as in Marr's model. Hippocampal-region damage in these network models is simulated by disabling the hippocampal-region module and observing the behavior of the remaining modules. These models can implement many aspects of associative learning, particularly classical conditioning, and they are useful for understanding how the hippocampal region may interact with the rest of the brain to facilitate certain kinds of learning [1].

2.3 Cortico-Hippocampal Interaction and Contextual Processing

Most of the computational models of Cortico-Hippocampal interaction in classical conditioning consider how conditioned stimuli were associated with responses and what role the hippocampal region might play in this association. But any conditioning experiment, indeed any form of learning, takes place against a background, or context, including the sights, sounds, and smells of the environment. There are also internal contextual cues such as motivation and drives. Typically, researchers try to minimize contextual cues or control for them by making sure that all subjects experience similar context. Nevertheless, it has long been recognized that context can and do affect what is learned.

From the early days of hippocampal research, it has been apparent that the hippocampal region plays an important role in contextual processing. Indeed, two early influential theories of hippocampal region function suggested that the region's chief function is contextual processing in general or processing spatial contexts in specific locations.

2.4 Cholinergic Modulation of Hippocampal-Region Function

Hippocampal-region processing is modulated by also other brain structures which provide neuromodulators, neurotransmitters and chemical messengers that affect how hippocampal-region neurons behave. Medial septum, a small group of cells that project to the hippocampus, is an important contributor to this neoromodulatory mechanism. Some of these cells produce the neurotransmitter acetylcholine (ACh), which are the cholinergic ones.

Normal hippocampal functioning critically depends on cholinergic input. If the septohippocampal cholinergic pathway is disrupted by either giving damage to the medial septum or by drugs that diminish ACh efficacy, hippocampal-region function is disrupted. Considering many studies, hippocampal-region disruption has qualitatively diverse effects on learning and memory behavior with respect to direct hippocampal region damage. Thus, effects of acetylcholine levels on learning memory should be considered [1].

2.5 Generalized Recirculation Algorithm

An algorithm called recirculation provided two important ideas that enabled backpropagation to be implemented in a more biologically plausible manner. The recirculation algorithm was subsequently generalized from the somewhat restricted case it could handle, resulting in the generalized recirculation algorithm or GeneRec which serves as a task-based learning algorithm.

GeneRec adopts the activation phases in the delta rule. In the minus phase, the outputs of the network represent the expectation or response of the network, as a function of the standard activation settling process in response to a given input pattern.

Then, in the plus phase, the environment is responsible for providing the outcome or target output activations. The $^+$ superscript is used to indicate plus-phase variables, and $^-$ to indicate minus-phase variables in the below equations and graphs.

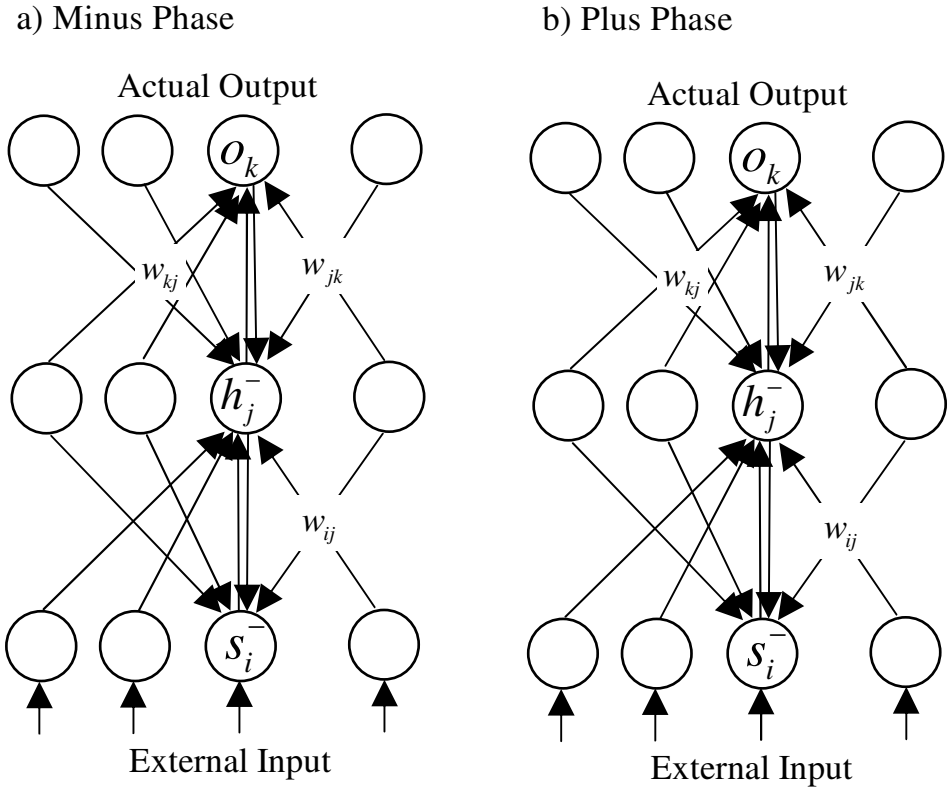

Figure 2.1 Illustration of the GeneRec algorithm, with bidirectional symmetric connectivity as shown. a) In the minus phase, external input is provided to the input units, and the network settles, with some record of the resulting minus phase activation states kept. b) In the plus phase, external input (target) is also applied to the output units in addition to the input units, and the network again settles [4].

It is important to emphasize that the full bidirectional propagation of information (bottom-up and top-down) occurs during the settling in each of these phases, with the only difference being whether the output units are updated from the network (in the minus phase) or are set to the external outcome/target values (in the plus phase). In particular, the hidden units need to receive the top-down activation from both the minus and plus phase output states to determine their contribution to the output error.

Conveniently, the learning rule is the same for all units in the network, and essentially just the delta rule:

$$\Delta w_{ij} = \varepsilon(y_j^+ - y_j^-)x_i^- \tag{2.1}$$

for a receiving unit with activation y_j and sending unit activation x_i in the phases as indicated (Figure 2.2). As usual, the rule adjusting the bias weights is just the same as for the regular weights, but with the sending unit activation set to 1:

$$\Delta \beta_j = \varepsilon(y_j^+ - y_j^-) \tag{2.2}$$

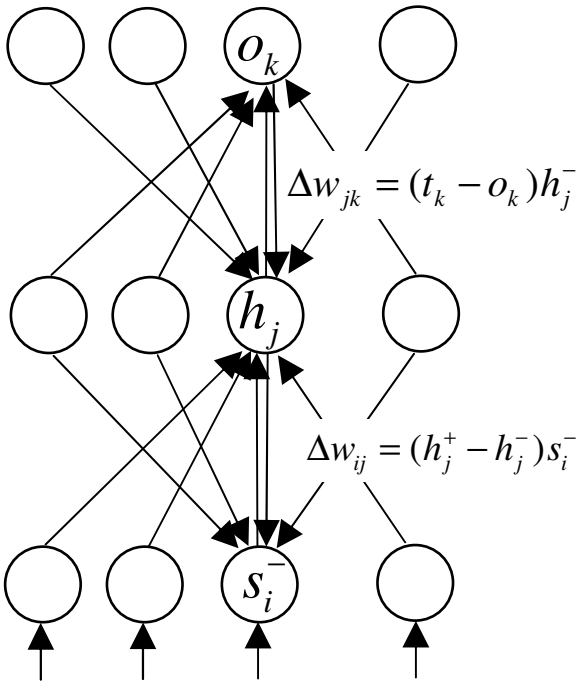

Figure 2.2 Weight updates computed for the GeneRec algorithm [4].

2.6 Soft Weight Bounding

The unbounded nature of error driven weights is incompatible with both the facts of biology and the point-neuron activation function which requires a separation between excitation and inhibition. Therefore, the following mechanism for bounding the error driven weights is used (noting, this does not apply to the bias weights, which have no such sign constraints):

$$\Delta w_{ik} = [\Delta_{ik}]_+ (1\text{-}w_{ik}) + [\Delta_{ik}]_- w_{ik} \tag{2.3}$$

where Δ_{ik} is the weight change computed by the error driven algorithm, and the $[x]_+$ operator returns x if $x > 0$ and 0 otherwise, while $[x]_-$ does the opposite, returning x if $x < 0$, and 0 otherwise.

If equation (2.3) is used iteratively, the weights approach the bounds of 1 and 0 exponentially slowly (softly). When there is a series of individual weight changes of equal magnitude but opposite sign, the weight will hover around 0.5, which corresponds well with the Hebbian interpretation of 0.5 as reflecting lack of positive or negative correlation. Similarly, as positive weight increases outweigh negative ones, the weight value increases proportionally, and likewise decreases proportionally for more negative changes than positive.

Weight bounding is appealing from a biological perspective because it is clear that synaptic efficacy has limits. We know that synapses do not change their sign, so they must be bounded at the lower end by zero. The upper bound is probably determined by such things as the maximal amount of neurotransmitter that can be released and the maximal density and alignment of postsynaptic receptors. What the soft weight bounding mechanism does is to assume that these natural bounds are approached exponentially slowly – such exponential curves are often found in natural systems. However, it is not known of any specific empirical evidence regarding the nature of synaptic bounding function [4].

2.7 Contrastive Hebbian Learning (CHL)

This algorithm is so named because it is the contrast (difference) between two Hebbian-like terms (the sender-receiver coproducts). The CHL algorithm traces its roots to the mean field or deterministic Boltzmann machine (DBM) learning algorithms, which also use locally available activation variables to perform error-driven learning in recurrently connected networks [3]. The DBM algorithm was derived originally for networks called Boltzmann machines that have noisy units whose activation states can be described by a probability distribution known as the Boltzmann distribution. In this probabilistic framework, learning amounts to reducing the distance between the two

probability distributions that arise in the minus and plus phases of settling in the network.

2.8 Inhibitory Interactions

Leak current and inhibition are two counterweights of neuronal activity. However, leak current is almost constant therefore, an important limitation. As a consequence it cannot respond to dynamic changes easily. Nevertheless, inhibition can play the role of a dynamic counterweight to excitatory input, as a function of the inhibitory input provided by the inhibitory interneurons known to exist in the cortex. General mechanism of these interneurons looks like sampling the general level of activation in the network.

The function of inhibitory interneurons could be compared to that of thermostat controlled air conditioner that prevents the network from getting too “hot” (active). Thermostat tries to maintain a roughly constant indoor temperature by their set point property even with varying levels of heat flux. The set point is provided with a negative feedback mechanism.

Two forms of connectivity involving the inhibitory interneurons and their connections with the principal excitatory neurons are present in the cortex which provides feedforward and feedback inhibition. A schematic representation of this is shown in Figure 2.3. Both type of connectivity are necessary and complement each other. Moreover, the inhibitory interneurons inhibit themselves by a negative feedback loop to control their own activity levels.

Feedforward inhibition occurs when the inhibitory interneurons in a hidden layer are driven directly by the inputs to that layer, and then send inhibition to the principal (excitatory) hidden layer neurons. This form of inhibition anticipates and counterbalances the excitation coming into a given layer from other layers.

Feedback inhibition occurs when the same layer that is being inhibited excites the inhibitory interneurons, producing a negative feedback loop. Thus, feedback inhibition reacts to the level of excitation from exploding (spreading uncontrollably to all units).

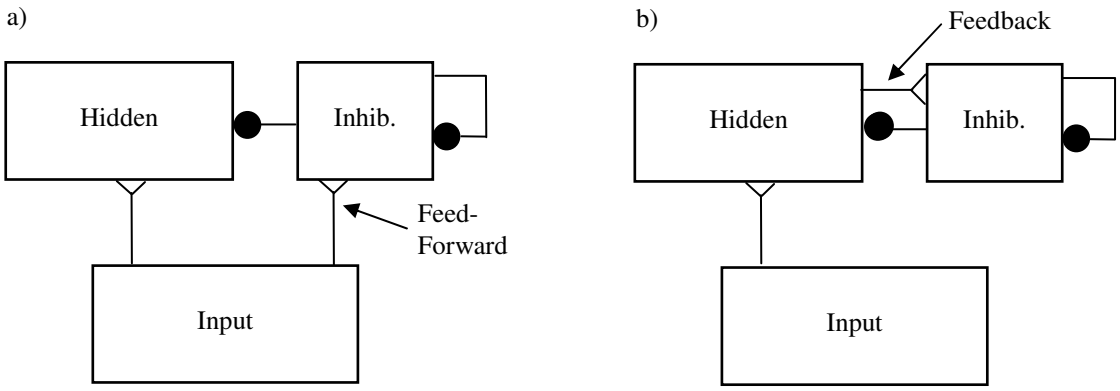

Figure 2.3 Two basic types of inhibitory connectivity (excitation is shown with the open triangular connections, and inhibition with the filled circular ones). a) Shows feedforward inhibition driven by the input layer activity, which anticipates and compensates for excitation coming into the layer. b) Shows feedback inhibition driven by the same layer that is being inhibited, which reacts to excitation within the layer. Inhibitory interneurons typically inhibit themselves as well [4].

To speed up and simplify simulations, the effects of inhibitory interneurons can be summarized by computing an inhibition function directly as a function of the amount of excitation in a layer, without the need to explicitly simulate the inhibitory interneurons themselves, as shown schematically in Figure 2.4. The simplest and most effective inhibition functions are two forms of a k-winners-take-all (kWTA) function. These functions impose a thermostat-like set point type of inhibition by ensuring that only k (or less) out of n total units in layer are allowed to be strongly active [4].

For a network that has unidirectional excitatory connectivity, that has no top-down connections for activity, feedforward inhibition is simple. Considering the bidirectional connectivity, feedforward inhibition should be understood better. The role of feedforward inhibition is to anticipate and counterbalance the level of excitatory input coming into a layer. Therefore, for a network that has a bidirectional excitatory connectivity, the inhibitory interneurons in the corresponding layer should also receive the top-down activity besides bottom-up activity.

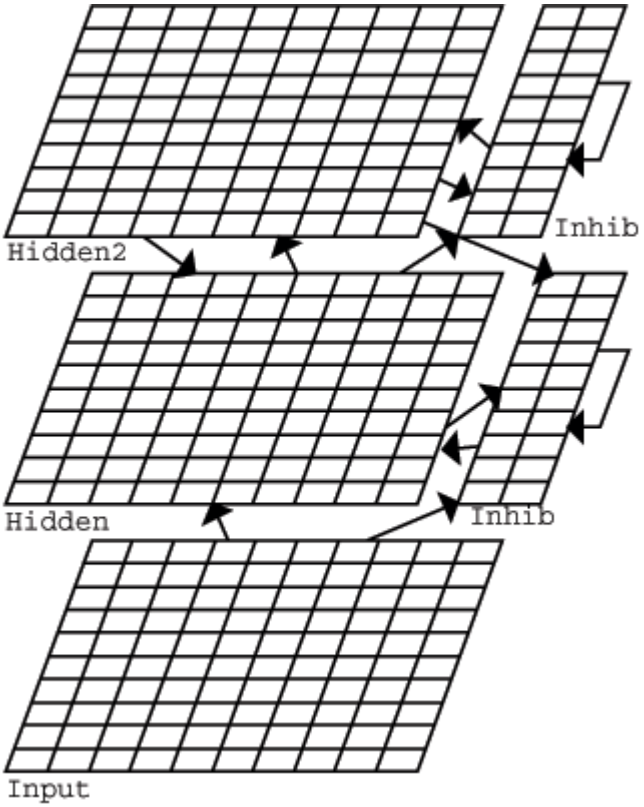

Figure 2.4 Inhibition network with bidirectional excitatory connectivity [4].

2.8.1 The k-Winners-Take-All (kWTA) Inhibitory Functions

The way kWTA functions operate to inhibit a layer of neurons is by not letting more than k active units out of n total in a layer. kWTA functions are also attractive from the biological perspective since it captures the set point property of the inhibitory interneurons by maintaining the activity level at a roughly constant level through negative feedback. On the other hand, in some particular applications this set point characteristic may become a weakness, if the model actually needs a more dynamic inhibition level for various inputs to the network. Nevertheless, a kWTA function enforces development of sparse distributed representations which can be beneficial from a functional perspective. Possible distributions of level of excitation across units in a layer, plotted on the Y axis, and rank order index on the X axis are shown in Figure 2.5. The basic kWTA function places the layer-wide inhibition value g_i between the k and k+1th most active units, as shown by the dotted lines.

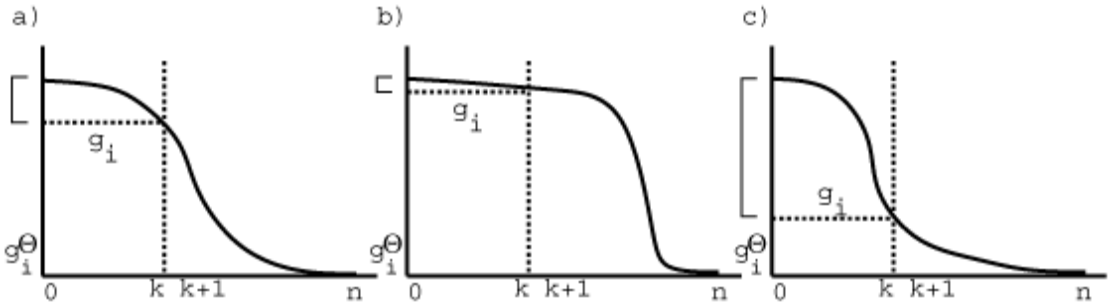

Figure 2.5 Possible distributions of level of excitation across units in a layer, plotted on the Y axis, and rank order index on the X axis. The basic kWTA function places the layer-wide inhibition value g_i between the k and k+1th most active units, as shown by the dotted lines. a) Shows a standard kind of distribution, where the most active units are reasonably above the inhibition. b) Has many strongly activated units below the threshold, resulting in a small excitatory-inhibitory differential for the most activated units. c) Has few strongly active units, resulting in a very large differential for the most activated units [4].

It is better to emphasize that the *k* units active in a kWTA function are the ones that are most active in their outputs. Thus, the first step in computing the kWTA functions is to sort the units according to their activations. Then a layer-wide level of inhibition is computed such that the top *k* units will have activity, while the rest will remain inactive. This inhibition value is then used by each unit in the layer when updating their activations.

2.9 The Cortico-Hippocampal Model of Gluck and Myers

Gluck and Myers approached hippocampal functioning top-down by beginning with a broad and abstract description of the computations that depend on the hippocampal region in classical conditioning. In their initial model, the hippocampal region was treated as an information-processing system that transformed stimulus representations according to specified rules within a series of linked connectionist networks. In particular, the model argued that the hippocampal region compresses (or makes more similar) the representations of inputs that co-occur or are otherwise redundant, and differentiates (or makes less similar) the representations of inputs that predict different future events. As a simple analogy, if thunder and lighting always co-occur, they should be treated as analogous and part of the same broader event. On the other hand, if two mushrooms look roughly alike but one is edible and one is poisonous,

then their representations should be made more distinct, exaggerating the subtle differences between them. The compressed and differentiated representations formed in the hippocampal region develop over multiple training trials through exposure to a range of stimuli and contextual regularities. These representations are then provided to other modules representing long-term storage in cerebral and cerebellar areas, which incorporate these new stimulus representations into their ongoing stimulus–response learning.

This information-processing theory is incorporated in the connectionist network model shown in Figure 2.6. Processing in the hippocampal region is implemented via a predictive autoencoder, which learns to transform stimulus inputs, through a narrow internal node layer, to outputs that reconstruct those inputs and also predict future reinforcement (or other salient events). Because the internal layer in this network contains fewer nodes than the input and output layers, the network is forced to compress redundant information while at the same time preserving and differentiating information that predicts reinforcement.

This hippocampal-region network then sends the new representations to a long-term memory (LTM) network, which models storage sites in the neocortex and cerebellum. A random recoding of the hippocampal-region network's internal-layer activations becomes the 'desired output' for the internal layer of the LTM network, and the error is the difference between this and the internal layer's actual output. The LTM network then uses an error-correcting rule to adapt its lower layer weights, just as it did to adapt its upper layer weights. Over time, the internal-layer nodes of the LTM network develop representations that are linear recombinations of those developed by the hippocampal-region network.

Within this model framework, broad hippocampal region damage is simulated by disabling the hippocampal region network (Figure 2.6). In this lesioned model, no new hippocampal-dependent representations are formed, and the training signal to the LTM network is silenced. The LTM network can adopt no new representations, although it can still learn to map from its existing representations to new behavioral responses [5].

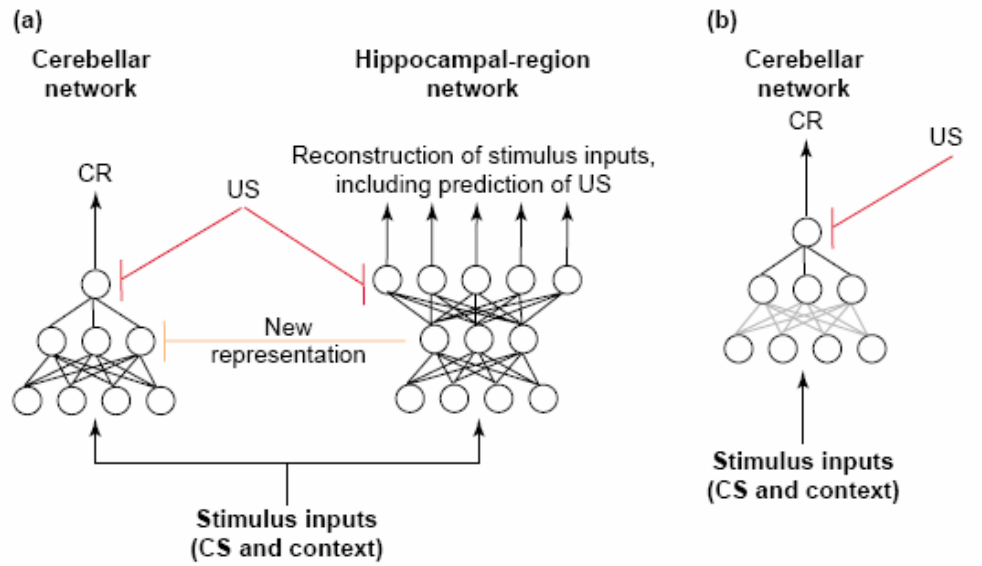

Figure 2.6 The cortico-hippocampal model (Gluck and Myers [1]). (a) In the intact model, the hippocampal region provides representational information to long-term memory sites, such as the cerebellum (illustrated here) and cortex; these representations are incorporated into ongoing learning to map from stimuli to responses. (CR, conditioned response; US, unconditioned stimulus) (b) After damage to the hippocampal region, the representational information is eliminated, although simple learning to map stimuli to responses is still possible [5].

3. SIMULATIONS AND RESULTS

3.1 Implementation of the Model

3.1.1 Hippocampal Network

The hippocampal network is a three-layer network with full connectivity between 18 input nodes and 40 internal nodes and 19 output nodes. The input consists of an 18-bit pattern I = (I1, I2,..., I18) representing the current values of five phasic cues and 13 tonic contextual cues. The desired output T = (T1, T2,..., T19) is the same 18-bit input pattern, as well as a 1-bit prediction of reinforcement. The network is trained by the GeneRec algorithm with kWTA inhibition. Node activations y are calculated as in the minus phase; an external input is provided to the input units, and the network settles. In the plus phase, external input (target) is also provided to the output units in addition to the input units, and the network again settles.

The weights are initialized according to a uniform distribution U(-0.1 to 0.1). They are then updated using

$$\Delta w_{ij} = \varepsilon(y_j^+ - y_j^-)x_i^- \tag{3.1}$$

The learning rate $\varepsilon = 0.25$ if $T_{19} = 1$ and $\varepsilon = 0.025$ if $T_{19} = 0$.

3.1.2 Cortical Network

The cortical network is a three-layer network with full connectivity between 18 input nodes and 60 internal nodes and a single output node. The input is the same as in the hippocampal network, whereas the desired output is the single bit T_{19} predicting reinforcement. Activation of nodes is computed as in the hippocampal network. The upper layer of weights, from hidden nodes j to output nodes k, is trained as in the hippocampal network, with learning rate $\varepsilon = 0.5$ if $T_{19} = 1$ and $\varepsilon = 0.05$ if $T_{19} = 0$. The weights and biases are initialized according to a uniform distribution U(-0.3 to 0.3).

Although the network algorithm is similar to hippocampal network; for the cortical network there are no external inhibitions.

The lower layer of cortical network weights is trained as

$$J = \sum v_{hj} x_h \tag{3.2}$$

where the *J* is the plus phase activation, v_{hj} is the connection strength from each hidden node *h* in the hippocampal network to hidden node *j* in the cortical network, x_h is unit activation of hidden nodes of the hippocampal network. These v_{hj} are nonadaptive and initialized according to a uniform distribution U(-1.0 to 1.0). This initialization and large number of internal layer nodes allows the lesioned model (cortical network only) to be able to solve random discriminations.

3.1.3 Stimuli and Training Schedule

Stimulus patterns are constructed by setting the first five bits to 0 or 1, depending on the presence or absence of five phasic cues. The next three bits code for a unique context: 101 for context X and 010 for context Y. The final 10 bits are a random string of 0s and 1s; constant across all stimulus patterns. They evolve slowly with time, so that on any trial there is some probability, P = 0.01, that one of the 10 bits will be inverted. This inversion is permanent unless randomly inverted back.

One block of training consists of a number of training trials, containing one presentation of each stimulus pattern being trained. These are intermixed with context-only presentations in a ratio of 1:20. For example, in the contextual conditional task (AX+, X-, AY-, and Y-), one block of training might consist of 10 presentations of context X, one presentation of phasic stimulus A in context X, 10 more presentations of X, 10 of context Y, one of A in Y, and 10 more presentations of Y. This ratio of context-only to training trials is about the minimum needed to ensure that background response to context alone remains low throughout training.

At the start of a simulation run, the network is initialized by training with 500 trials in which the input vector and output are both set to 0. This initialization ensures that the network has a low baseline output rate in the absence of input.

3.2 Progress of Code Implementation

Before implementing Gluck and Myers Cortico-Hippocampal model with GeneRec, their code was re-written for test purposes. We tried to use the training data based upon their original work so that we could compare the results of their original implementation with backpropagation with our implementation based on the GeneRec algorithm. Therefore, we first developed a code for generating the training data. The following simulations used the output of this code as the training data. The data set mostly used for testing the performance of the networks individually and as system was the stimulus pattern: [X-, AX+, X-, Y-, AY-, Y-] with [10, 1, 10, 10, 1, 10] number of presentations, respectively. Here, X or Y represent static context, A represents one of the five phasic cues and – and + signs represent unconditioned stimulus, which is the training output value for the cortico-cerebellar network.

The hippocampal network and cortico-cerebellar network codes were written individually at first and then combined together to get the cortico-hippocampal network. The performance of the model is shown in Figure 3.1.

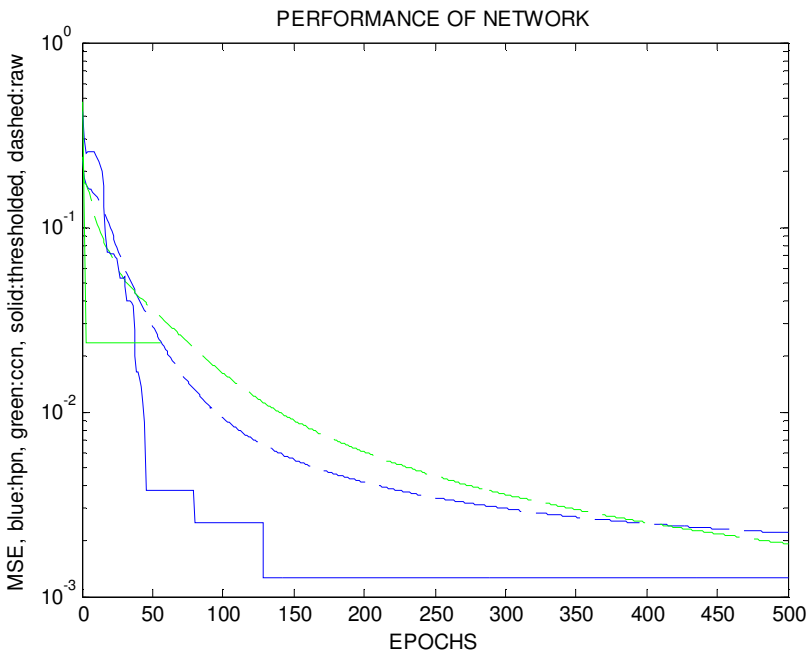

Figure 3.1 Mean squared error performance of Gluck and Myers intact cortico-hippocampal model.

The next step was to write the code with GeneRec for the hippocampal network. Note that in this case, there are no weight bounding and no inhibition. The performance of the model is presented in Figure 3.2.

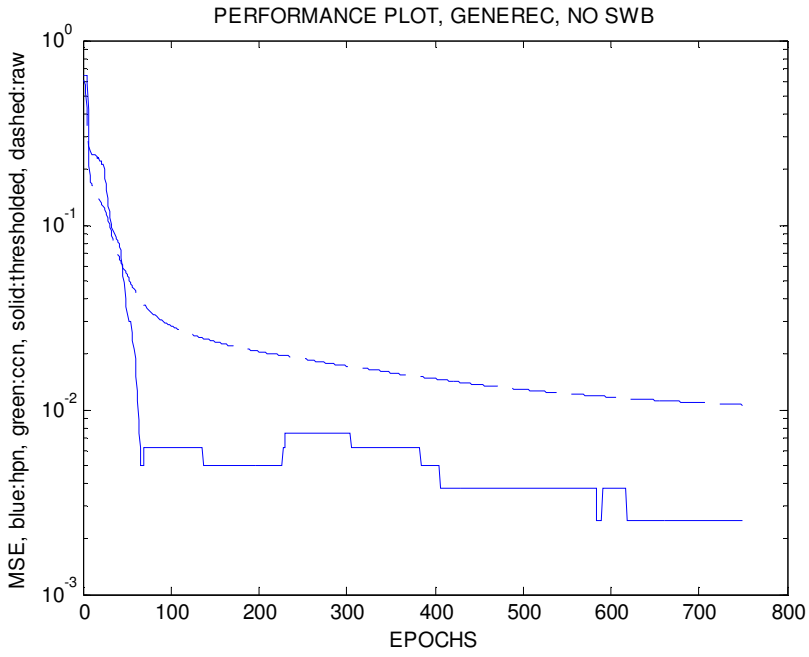

Figure 3.2 Mean squared error performance of GeneRec implementation of Gluck and Myers hippocampal network. No soft weight bounding. No inhibition.

To add soft weight bounding (SWB) to the code, it was first tested with -1, 1 boundaries. The performance of the system was poor as compared to the system with no SWB, as shown in Figure 3.3.

kWTA inhibition was added next to the hippocampal network with SWB. The k values for the output layer and the hidden layer were calculated dynamically for every input. The mean squared error performance of GeneRec implementation of the hippocampal network with -1, 1 soft weight bounding (SWB) and with kWTA inhibition is shown in Figure 3.4. The number of neurons in the hidden layer was 40.

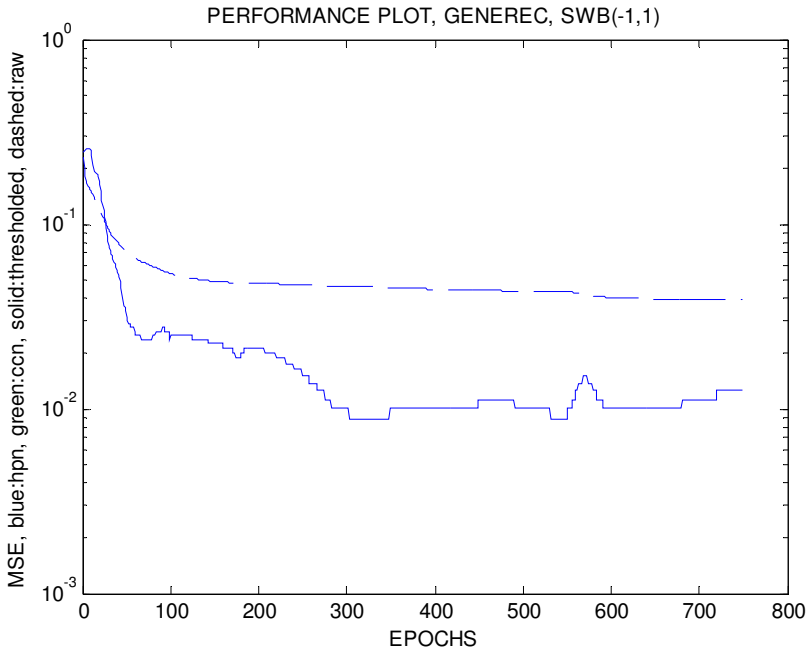

Figure 3.3 Mean squared error performance of GeneRec implementation of Gluck and Myers hippocampal network with -1, 1 soft weight bounding (SWB). No inhibition.

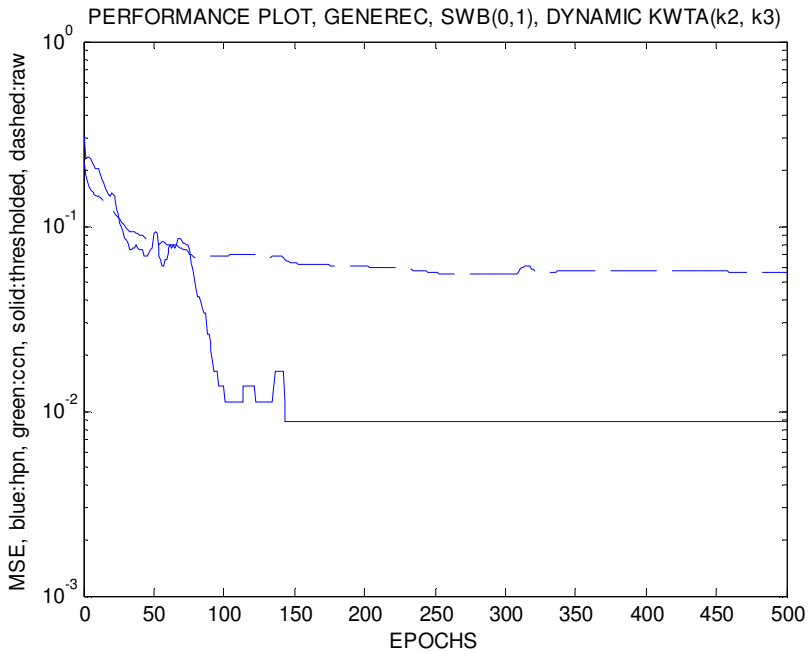

Figure 3.4 Mean squared error performance of GeneRec implementation of Gluck and Myers hippocampal network with -1, 1 soft weight bounding (SWB) and with kWTA inhibition. Number of neurons in the hidden layer is 40.

Next, different numbers of hidden layer neurons were tested, such as 10, 20, and 30. Figure 3.5 shows the mean squared error performance of GeneRec implementation of Gluck and Myers hippocampal network with -1, 1 soft weight bounding (SWB) and with kWTA inhibition; the number of neurons in the hidden layer was 20.

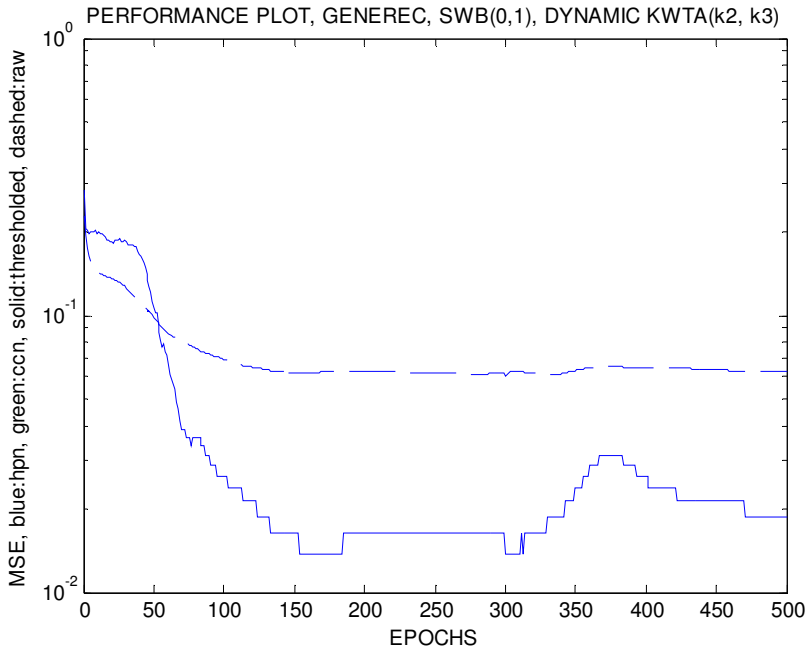

Figure 3.5 Mean squared error performance of GeneRec implementation of Gluck and Myers hippocampal network with -1, 1 soft weight bounding (SWB) and with kWTA inhibition. Number of neurons in the hidden layer is 20.

After having satisfactory results with the hippocampal network, code was written for the cortico-cerebellar network. Individual performance of the cortico-cerebellar network was satisfactory with no inhibition. kWTA inhibition was not suitable for this network, since it has only one output layer neuron. Therefore, inhibitory interneurons were added to the cortico-cerebellar network. However, performance of the network was dramatically unsatisfactory.

When these two networks combined as a system, the performance was totally unsatisfactory (Figure 3.6).

As the results suggested, to have a working model the only way was to abandon inhibition and weight bounding for the cortico-cerebellar network. Figure 3.7 shows our

simulation results for an example with no inhibition and with no soft weight bounding for the cortico-cerebellar network.

The results were far from convergence with even with no inhibition and no weight bounding for the cortico-cerebellar network. Indeed there were signs of convergence on the performance plots, if we do not consider the huge oscillations.

Perhaps the best thing to do is to abandon soft weight bounding and inhibition for both networks. For the following simulations there are no SWB and no inhibition (Figure 3.8 and 3.9).

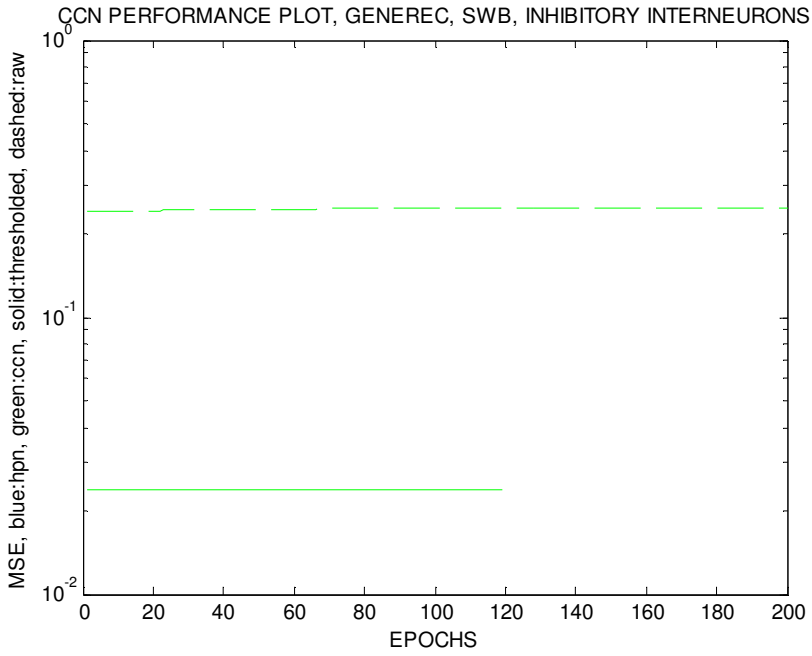

Figure 3.6 Mean squared error performance of GeneRec implementation of Gluck and Myers cortico-cerebellar network with -1, 1 soft weight bounding (SWB) and with kWTA inhibition. Number of neurons in the hidden layer is 60.

3.3 Representational Differentiation

Hippocampal-region network forms representations and these representations are subject to two biases. One of them is to compress the representations of stimuli that are redundant and the other is to differentiate the representations of stimuli that predict different outcomes. Each of these biases can be used to explain data in intact and HR-

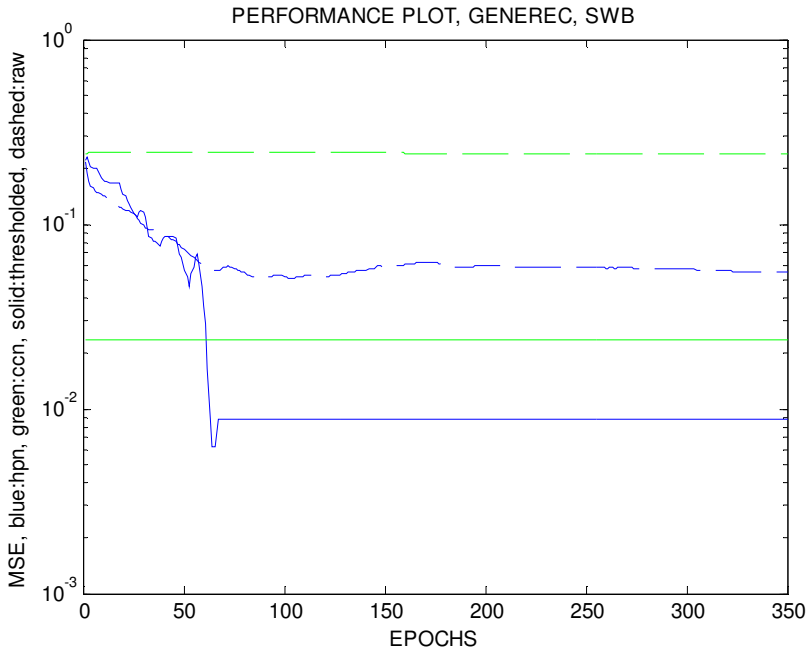

Figure 3.7 Mean squared error performance of GeneRec implementation of Gluck and Myers intact cortico-hippocampal model with both sub-networks having inhibition and soft weight bounding.

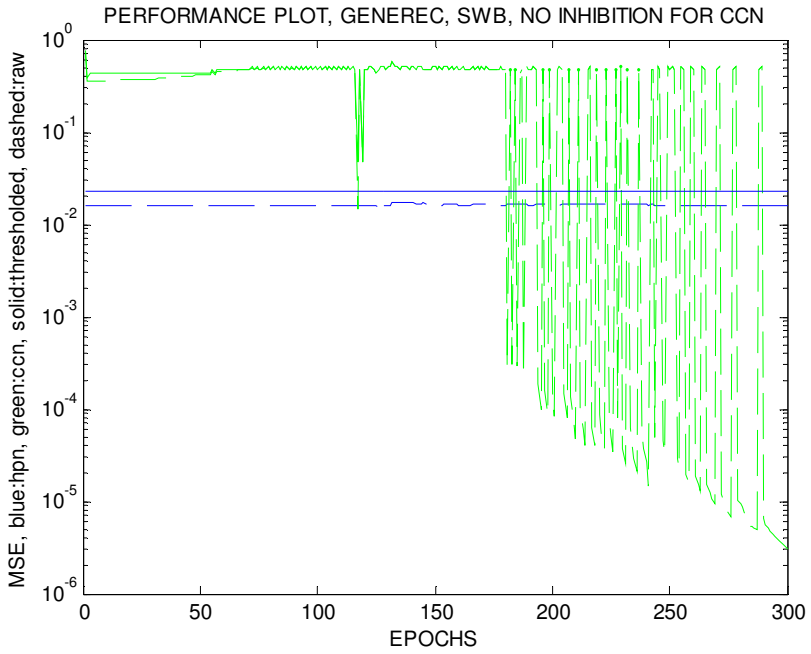

Figure 3.8 Mean squared error performance of GeneRec implementation of Gluck and Myers intact cortico-hippocampal model with cortico-cerebellar network having no inhibition and no soft weight bounding.

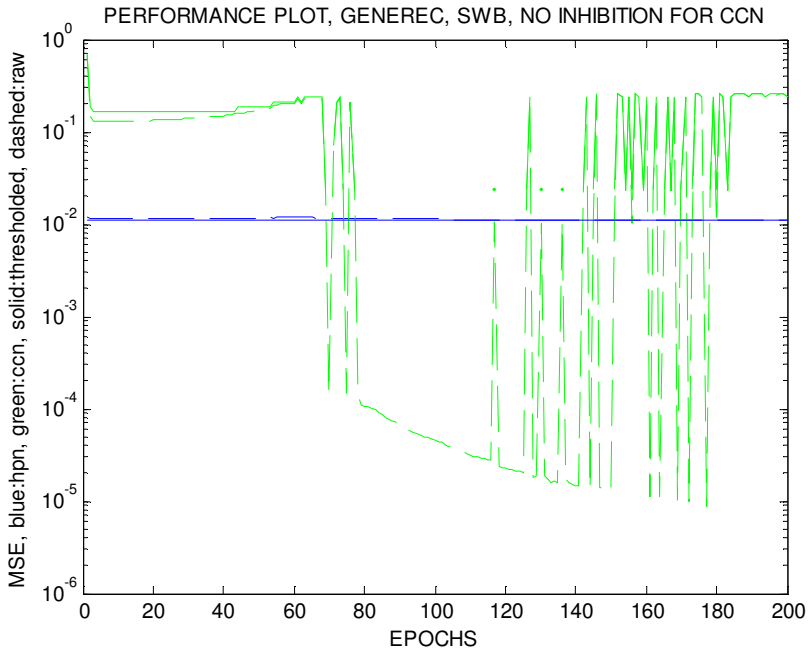

Figure 3.9 Mean squared error performance of GeneRec implementation of Gluck and Myers intact cortico-hippocampal model with cortico-cerebellar network having no inhibition and no soft weight bounding.

lesioned animals. Below are several examples of learning behaviors that appear to involve representational differentiation.

3.3.1 Acquisition

The most rudimentary eyeblink conditioning task is acquisition: learning to respond to a cue that has been paired with the US. The Rescorla-Wagner model captures this behavior, suggesting that the cerebellum alone should be sufficient to mediate conditioned acquisition and hence learning should not be disrupted by HR lesion. Indeed, acquisition of a conditioned eyeblink response is not disrupted by HR lesion in humans, rabbits, or rats.

Conditioned acquisition is simulated in the intact cortico-hippocampal model by presenting a series of training trials. First, the model is given trials consisting of just the experimental context – call it X – a series of inputs meant to present the background

sights, smells, and sounds of the experimental setup; the model learns not to give a conditioned response to the context alone. These trials correspond to the time spent acclimating an animal to the experimental chamber, before any explicit training begins, a standard procedure in experimental studies of animal conditioning.

Next comes the actual acquisition training. Because the training takes place in context X, learning to respond to a light CS can be redefined as learning to respond to light-in-X but not to the context alone X-. With enough training, the model learns to respond when the light is present but not to the context alone. With training, internal-layer representation in the cortico-cerebellar network changes by copying the representations in the hippocampal-region network.

However, this new differentiated representation is probably not necessary to acquire a conditioned response to a single light CS. The task is so simple that just about any random recoding in the lower layer of cortico-cerebellar network weights is probably sufficient. As long as there is at least one node in the internal layer that gives a different response to light-in-X and X alone, that node can be used to drive the presence or absence of a CR.

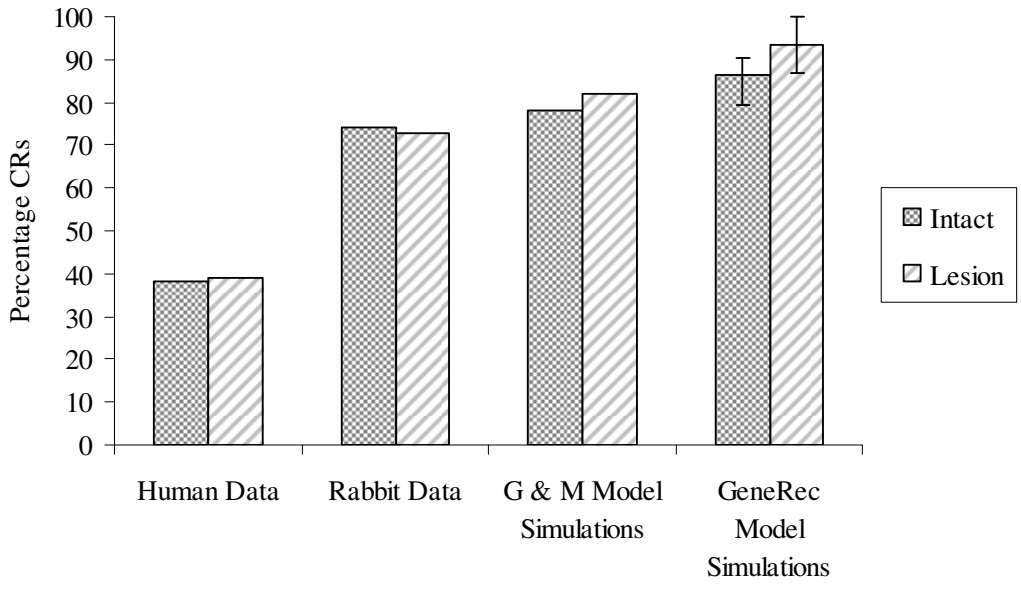

Figure 3.10 Conditioned acquisition, learning that a tone CS predicts an airpuff US, is not disrupted by hippocampal-region damage.

3.3.2 Discrimination and Reversal

Simple discrimination involves learning that one CS(light+) predicts the US while a second CS(tone-) does not. This means that conditioned responses should follow light+ but not tone-. In general, discrimination learning in the eyeblink-conditioning paradigm is not disrupted by hippocampal-region damage. Similarly, hippocampal-region damage generally does not impair a range of discrimination tasks in animals, including discrimination of odors, objects, textures and sounds.

In the intact cortico-hippocampal model, the hippocampal-region network constructs new representations that differentiate light+ and tone-, facilitating the mapping of light+ to one response and tone- to another. However, the discrimination task is so simple that such representational changes are probably not necessary; any random initial representations in the cortico-cerebellar network are probably different enough to allow mapping to different responses. Thus, the HR-lesioned model should be able to learn a conditioned discrimination.

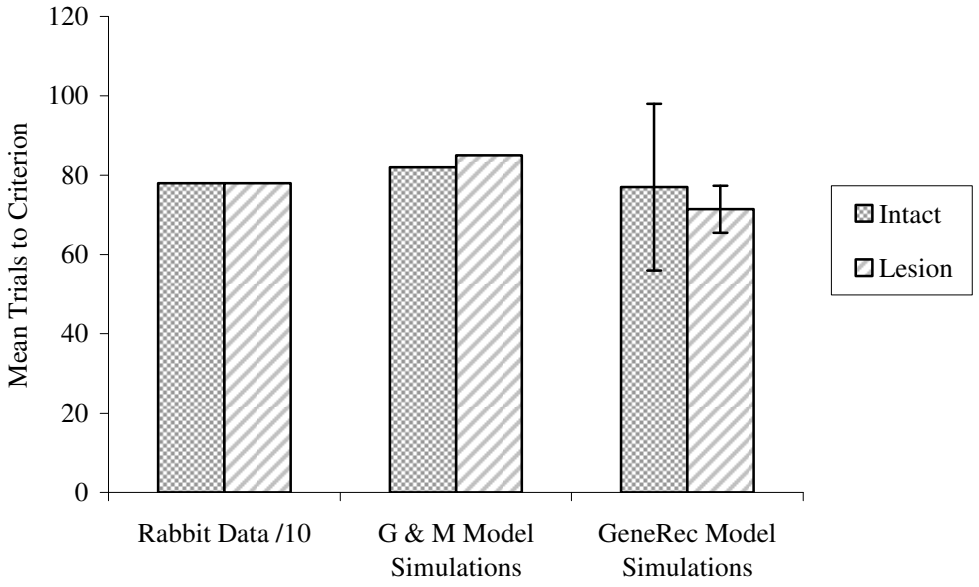

Figure 3.11 Conditioned discrimination: learning to respond to one CS light+, which is paired with the US, but not to CS tone-, which is not paired with the US.

Empirical data have often been interpreted as arguing that conditioned discrimination is hippocampal-independent. Gluck and Myers offer a different

interpretation: The hippocampal-region may not be strictly necessary for some kinds of learning; but when it is present, it normally contributes to all learning. Even in a simple task such as discrimination (or acquisition), where a priori representations probably suffice to allow learning, the hippocampal region is constantly forming new stimulus representations that compress redundant information while differentiating predictive information, whether these new representations are needed or not [1].

However, the usefulness of this hippocampal participation becomes apparent if task demands change. For example, suppose the discrimination is reversed so that after learning to respond to light+ but not tone-, the contingencies are reversed, so tone+ now begins to predict the US and light- does not. In the intact model, the hippocampal region network has already done the work of differentiating the representations of light and tone; once the contingencies are reversed, all that needs to be done is to map those representations to new responses. In the lesioned model, the situation is quite different: the representations of light and tone are fixed, and so they are not differentiated during the original discrimination. Thus, the reversal requires first unlearning the original discrimination and then learning the reversed discrimination. This process may be quite lengthy in comparison to reversal in the intact model. In rabbit eyeblink conditioning, several studies show that hippocampal-region damage disrupts discrimination reversal.

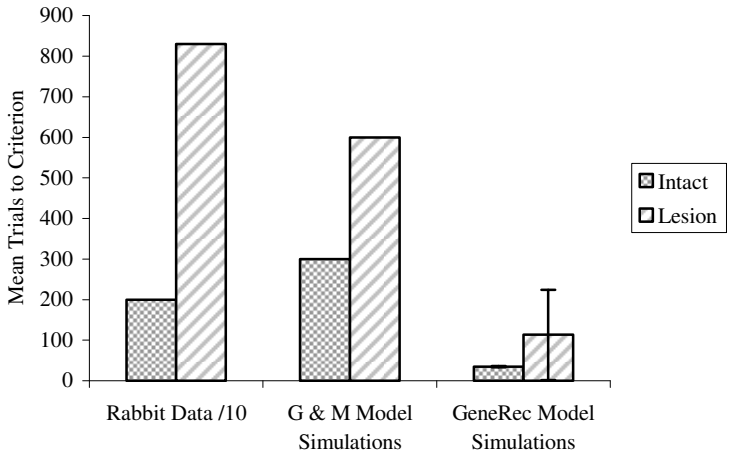

Figure 3.12 Discrimination reversal.

3.4 Representational Compression

Just as the hippocampal region is assumed to differentiate the representations of stimuli that should be mapped to different responses, the hippocampal region is assumed to compress the representations of stimuli that co-occur and should be mapped to similar responses. Behaviors that reflect representational compression should be disrupted after hippocampal-region damage.

3.4.1 Sensory Preconditioning

Sensory preconditioning involves unreinforced exposure to a compound of two stimuli (tone&light- exposure), followed by light-US pairings (light+ training). The associations learned to the light should partially transfer to tone, as a result of the paired exposure. Hippocampal-region damage (specifically fimbrial lesion) abolishes sensory preconditioning in the rabbit eyeblink preparation. In the intact model, tone&light- exposure results in compression of the representations of tone and light, since both stimuli co-occur and neither predicts the US or any other salient event. Subsequent associations to light partially activate the representation of tone, and learning transfers. In the lesioned model, there are no representational changes during the exposure phase, and as long as light and tone are distinct stimuli that activate different (fixed) representations, there is little chance that associations made to light will transfer to tone.

3.4.2 Learned Irrelevance

Another behavior involving representational compression is learned irrelevance. The paradigm is schematized in Table 3.1. In phase 1, subjects in the exposed group are given presentations of a CS (e.g., light) and a US, uncorrelated with each other. Subjects in the non-exposed group are given equivalent time in the experimental context but receive no presentations of light or the US. In phase 2, all subjects receive light-US pairings. Subjects in the exposed group are much slower to learn the light-US association.

In the intact cortico-hippocampal model, phase 1 exposure to a CS (e.g., light) and a US causes representational changes. The representation of the light becomes compressed, together with the representations of the background contextual cues, since

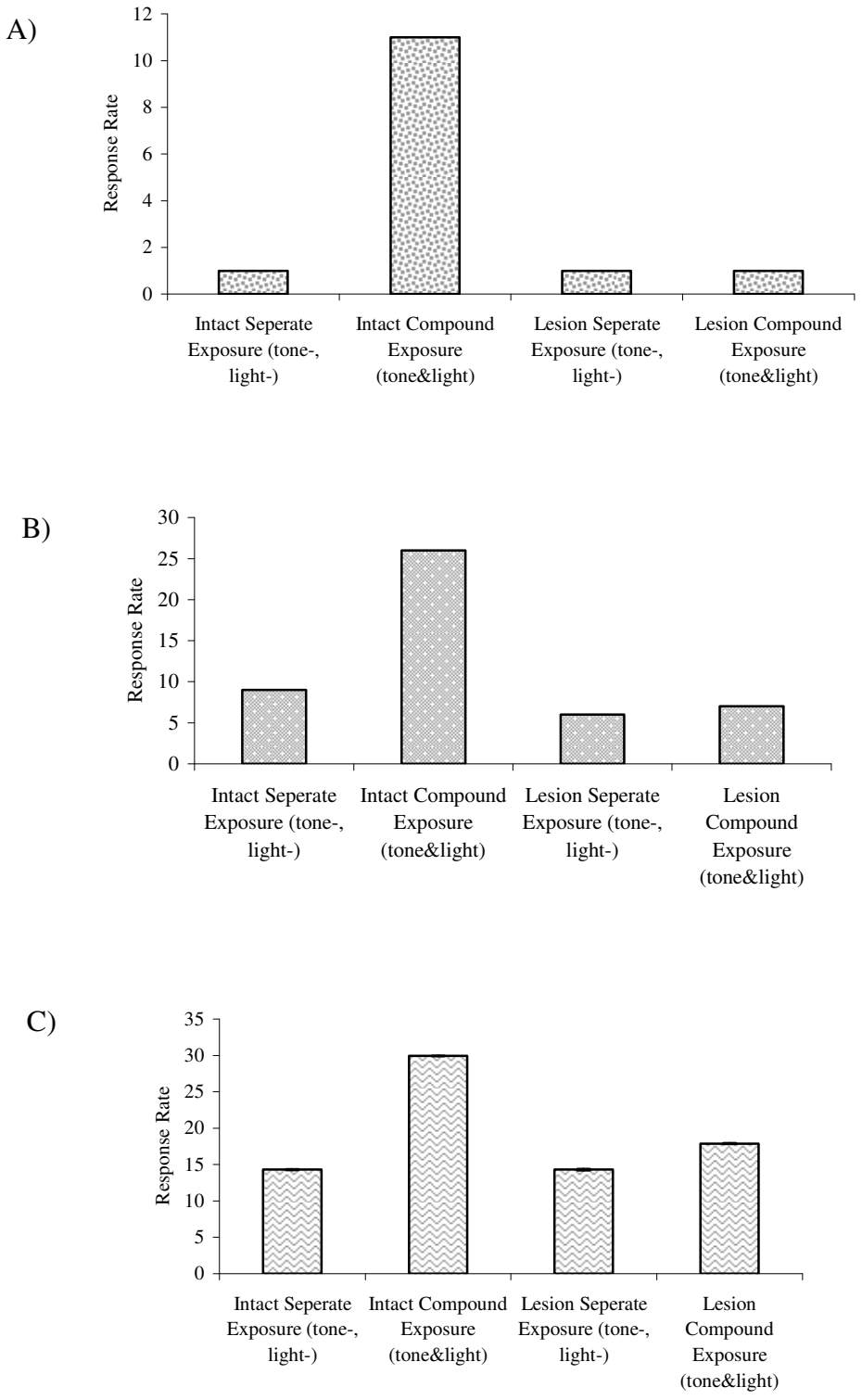

Figure 3.13 Sensory Preconditioning. A: Rabbit data. B: Gluck & Myers model simulations. C: GeneRec model simulations.

Table 3.1
The Learned Irrelevance Paradigm.

Group	Phase 1	Phase 2
CS/US exposure	Light and airpuff (uncorrelated)	Light → airpuff ….SLOW!
Sit exposure	Animal sits in experimental chamber	Light → airpuff ….normal speed

neither predicts the US well. In effect, the light is treated as a sometimes-occurring aspect of the context, one that is of no use in predicting US arrival. This representational compression of light and context will hinder phase 2 learning to respond to the light but not the context alone. Thus, there is a learned irrelevance effect in the intact cortico-hippocampal model. Since learned irrelevance is interpreted in terms of representational compression, it is not shown in the HR-lesion model.

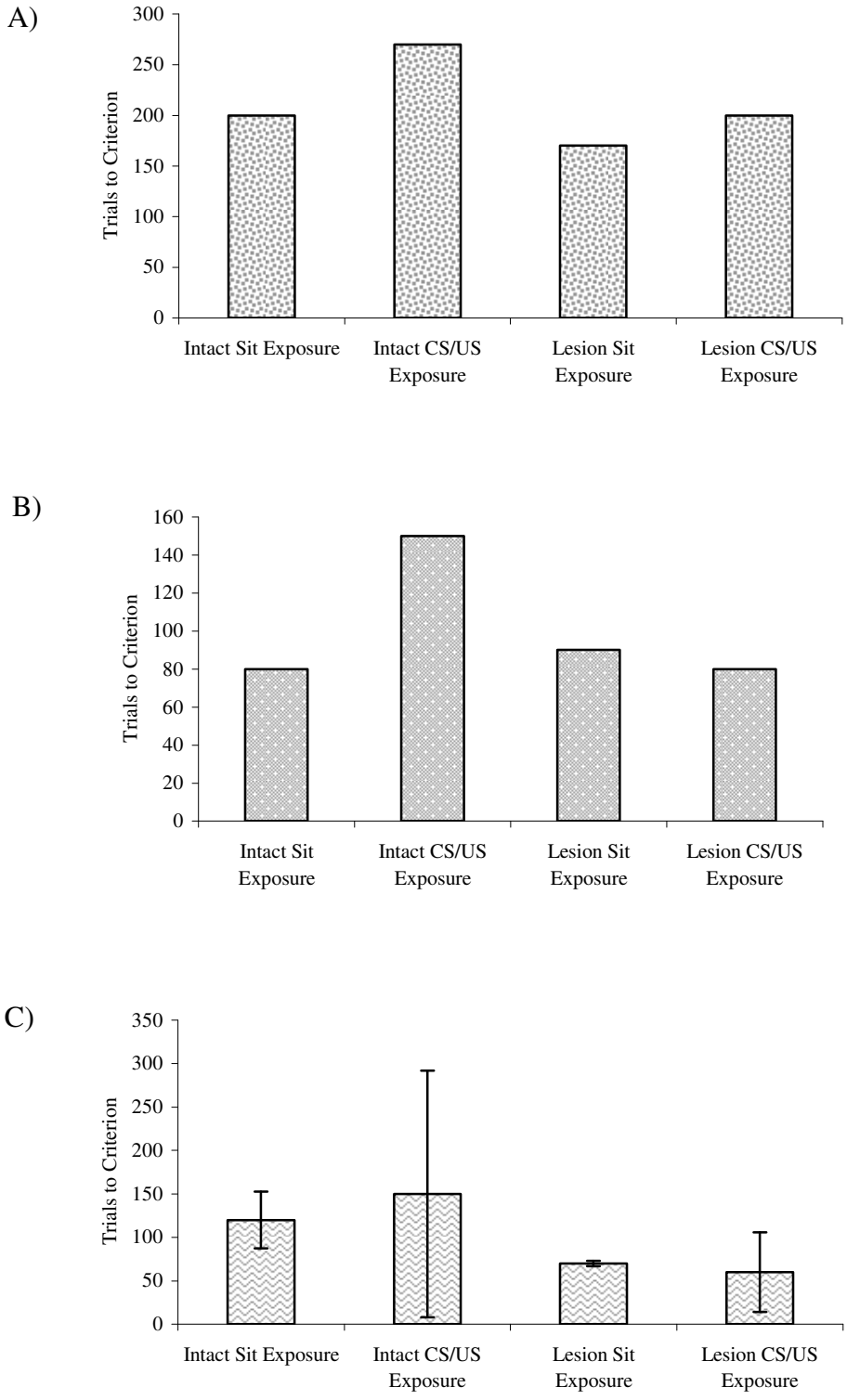

Figure 3.14 Learned irrelevance and HR-lesion. A: Rabbit data. B: Gluck & Myers model simulations. C: GeneRec model simulations.

4. DISCUSSION

4.1 Introduction

In this thesis all the training data, network model and several other data or principles are based on Gluck and Myers' cortico-hippocampal model. Where possible all the data and variables were used with their original values. Gluck and Myers published many articles on their model and also a book: "Gateway to Memory: An Introduction to Neural Network Modeling of the Hippocampus and Learning" [1]. The book was somewhat a review and a broad collection of their up to date work on their cortico-hippocampal model. Therefore, data and simulations performed are based on this book.

The GeneRec algorithm was chosen as a learning mechanism for this work for several reasons. First of all, the objective of this work was to come up with a network model of the hippocampus and related regions that is similar to biological networks and functioning similar to biological networks in its learning mechanism. The GeneRec algorithm in many ways seems more biologically plausible rather than the backpropagation algorithm that was used by Gluck and Myers originally. Moreover, it has a solid theoretical background and there are many publications using this algorithm as their learning rule.

4.2 Training Data

As a first step in this work a computer code was written to generate the training data, based on Gluck and Myers cortico-hippocampal model's training data principles. Training data set for any case included 18-bits of input vector, 19-bits of target output vector for hippocampal network and 1-bit of target output for cortico-cerebellar network. All the data were formed of 0s and 1s. The input vector includes stimulus patterns, tonic cues, phasic cues and random formed constant contextual bits which evolve slowly by time. Further details are in the "Stimuli and Training Schedule" part above in the simulation details.

As the second step, Gluck and Myers' model was realized using Matlab. First the hippocampal network was written and tested, and then cortico-cerebellar network was written and tested. Finally both networks were combined. With several trials, performance and results of the network were as in their original work. This reproduction work of the model was very valuable. The hippocampal network was a typical three layer network with an important difference which is, trying to reproduce input vector at the output and also trying to guess unconditioned stimulus. The cortico-cerebellar network was again a three layer network with 18 input nodes but only one output node to guess the unconditioned stimulus. While the hippocampal network used backpropagation for learning, the cortico-cerebellar network did not. For the intact model both networks work together. Hidden layer outputs of the hippocampal network are used to obtain the target output values of the hidden layer nodes of the cortico-cerebellar network. For the lesioned model there are no teaching signals for the hidden layer nodes of the cortico-cerebellar network.

4.3 The GeneRec Algorithm

When it came to write the code with GeneRec, it was the same strategy as before; divide the network into two as hippocampal network and cortico-cerebellar network. The hippocampal network was performing well with GeneRec. However, our concern here was to write a code that is as much as similar to the biological constraints. That is, weights should not take negative values and they should be limited in their values. The method used at this stage was SWB (Soft Weight Bounding). The SWB function modifies weight changes during weight update. For any given lower and upper limits, weight change exponentially slows through the limits. Weight bounding is appealing from a biological perspective because it is clear that synaptic efficacy has limits. However, we do not know of any specific empirical evidence regarding the nature of synaptic bounding function.

There were two alternatives for applying SWB. The first was to apply it with a negative lower value. Therefore, the boundaries would be such as (-1, 1). The second was to apply it with 0 lower boundary. However, in such a case there should be an inhibitory mechanism to prevent over-excitation, which in turn prevents convergence.

First, SWB with (-1, 1) boundaries is applied. Addition of SWB increased required number of epochs for the network's learning criterion but did not have a significant effect on convergence. To apply SWB with 0 lower boundary kWTA inhibition is planned to be added to the network.

4.3.1 kWTA

As mentioned above in Section .2.8, a kWTA function ensures that no more than k units out of n total in a layer are active at any given time. Actually this function produces similar results to inhibitory interneuron activity. Also it is computationally less resource intensive considering inhibitory neurons. Moreover, it is easier to apply since it does not require addition of extra network layers and is more manageable with less parameter. However, it has some drawbacks; for example, the activity level for the applied network layer is roughly constant. Since it has different types as basic or average based kWTA, deciding on any type may require experience or lots of trials.

Another challenge with kWTA is to decide, in any layer, how many nodes will be active at any time. For the output layer of the hippocampal network the case was tricky. Since the hippocampal network is thought as an auto-encoder, the number of active nodes in the output layer would be roughly the same as in the input nodes. However, for the hidden layer of the hippocampal network, finding the working k value was not easy. Furthermore, any fixed k value would not be biologically meaningful. After excessive trials the best way appeared to be calculating the k value related to the overall activity distribution of nodes. Considering a normal statistical probability distribution, the k value is calculated from the number of most active units which their z value is bigger than 0.9.

The next step with the kWTA was to investigate the performance of the system with different number of hidden layer nodes. In the original Gluck and Myers cortico-hippocampal model, the number of internal layers for the hippocampal network is 10. For the first trials, the number of hidden layer nodes was 40 for our model with kWTA inhibition. Less hidden layer nodes were tried with different z values; however the performance of the system decreased significantly with fewer nodes. Therefore, in the

following simulations the number of hidden layer nodes for hippocampal network was taken as 40.

At the last step, SWB is applied with (0, 1) boundaries and with kWTA inhibition for the hippocampal network. Although convergence could be achieved with this modification, mean squared errors increased. This was not surprising because the network was limited in its weights from 0 to 1. Also, even it performs well it was not easy to get 0s or 1s as outputs. Reaching to weight boundaries is exponentially hard with SWB. Therefore, mse (mean squared error) values were also calculated with a threshold (0.5). For the thresholded mse values; if the output of any neuron is bigger than 0.5 it is calculated as 1 and if it is less than 0.5 then it is calculated as 0.

4.4 The Cortico-Cerebellar Network

Having satisfactory results with the GeneRec for the hippocampal network, it was time to apply GeneRec to the cortico-cerebellar network. As it is mentioned above the cortico-cerebellar network was different in its learning mechanism. The teaching signal for the hidden layer nodes were not through the output layer activity of the cortico-cerebellar network but a function of the hidden layer activity of the hippocampal network.

For the j^{th} neuron in the hidden layer of the cortico-cerebellar network target plus phase value is calculated as,

$$J = \sum v_{hj}\, x_h \tag{4.1}$$

where the v_{hj} are connection strengths from each hidden node h in the hippocampal network to hidden node j in the cortical network. Further details are above in the simulations part.

4.4.1 Inhibition

During the development of the cortico-cerebellar network code we faced various problems. As mentioned before, when using kWTA inhibition, how to compute the k values for each layer must be decided. For the hidden layer of the cortico-cerebellar

network any k value may be used, leading to a great number of active hidden layer nodes. Another approach is to use a similar value to the one used in the hippocampal network. However, the real challenge is to decide the k value of the output layer, since there is only one node. The k value is an integer; for the output layer there are only two choices will it be 0 or 1. If it is 1, the network always computes 1. If it is 0, the network always computes 0 which is also wrong. There is no way of predicting the k value. Even if there is, actually it is the function of network to guess the output will be 0 or 1. As a consequence it was apparent that another method of inhibition should have been used.

The second method of inhibition choice for the cortico-cerebellar network was the use of inhibitory interneurons as it is been in a biological network. These inhibitory interneurons were added as two additional layers interacting with all three layers of the cortico-cerebellar network. They had feedforward, feedback and self inhibitory connections. Addition of these neurons and thereby new layers required addition of many parameters such as relative strengths of feedforward, feedback and self inhibitory connections for even each layer level. However, nonlinearity of the system with its highly dynamic character and sophisticated structure dramatically decreased its manageability. After excessive trials, the performance of the network was not found satisfactory; even not better than the kWTA inhibition application for the cortico-cerebellar network. Consequently, the only choice was to abandon inhibition for the cortico-cerebellar network.

4.5 Cortico-Cerebellar and Hippocampal Networks

Finally two networks, that are cortico-cerebellar and hippocampal networks were combined as a system. However, it was hard to state whether the cortico-cerebellar output converged. Although, the minimum values of the mean squared error trend was looked like it is converging, actually it was oscillating but not converging. The reason of these huge oscillations was rapid representational changes in the hidden layer of the hippocampal network. To prevent these rapid representational changes, kWTA inhibition and therefore SWB was abandoned. Consequently, the network as system, started to give meaningful results.

4.6 Representational Compression

For the acquisition case which is learning to respond to a cue that has been paired with the US, the comparison is made on percentage conditioned responses after a fixed number of epochs. The comparison data was obtained from the Gluck and Myers' articles and consisted of data gathered from humans and rabbits. Both subjects responded at nearly same percentage for their lesioned and intact ones. Human subjects responded around 40% for both with lesioned and intact hippocampus. Rabbits responded around 70%. Model simulations of Gluck and Myers resulted in 78% for the control group (intact) and 82% for the amnesic group (lesion). GeneRec implementation of Gluck and Myers model gave 86% for the control group (intact) and 93% for the amnesic group (lesion). Standard deviation for our implementation was 4.1 for the control group and 6.6 for the amnesic group.

The next example was another representational differentiation case: discrimination and discrimination reversal. For this case, comparison is made on mean trials to criterion. For this case we only had rabbit data which is again from Gluck and Myers articles. The first comparison is made on discrimination. Rabbits responded around 80% for both the control group and the amnesic group. Model simulations of Gluck and Myers resulted in 82% for the control group (intact) and 85% for the amnesic group (lesion). The GeneRec implementation of Gluck and Myers model gave 77% for the control group (intact) and 71% for the amnesic group (lesion). The standard deviation for our implementation was 21 for the control group and 5.9 for the amnesic group.

The next comparison is made on discrimination reversal. Control group rabbits reached the criterion around after 2000 trials where amnesic group of rabbits reached the criterion around after 8300 trials. Model simulations of Gluck and Myers resulted in 300 epochs to reach the criterion for the control group and 600 epochs for the amnesic group. The GeneRec implementation of Gluck and Myers model gave 35 epochs for the control group (intact) and 113 epochs for the amnesic group (lesion). The standard deviation for our implementation was 1.5 for the control group and 111 for the amnesic group. The huge standard deviation of the amnesic group considering the mean value was significant. This was due to the wide range of results for this case. Actually half of

the simulations reached the criterion after many epochs yet some simulations reached the criterion as fast as the control group. This was possibly due to initial representations formed after discrimination before reversal simulations ran. These initial representations, by chance allowed the cortico-cerebellar network to form new and correct representations just by changing its output layer weights.

4.7 Representational Differentiation

We had two representational differentiation cases. One is sensory preconditioning and the other is learned irrelevance. For the sensory preconditioning case, comparisons are based upon response rates. Experimental data was from rabbits. Sensory preconditioning involves unreinforced exposure to a compound of two stimuli (tone and light- exposure), followed by light-US pairings (light+ training). Rabbits gave very low response rates to separate exposure for both control and amnesic group. On the other hand, the control group rabbits gave significantly higher, even more than 10 times, response rates to compound exposure when both stimuli were given simultaneously. The Gluck and Myers model simulations gave similar results. Response rates for separate exposure and compound exposure for amnesic group were around 15, where compound exposure to control group gave response rates around 25. The GeneRec implementation of Gluck and Myers model was similar considering the results of the original implementation. Separate exposure to stimuli yielded a response rate of 14 for both the control and the amnesic group. Compound exposure yielded a response rate of 18 for the amnesic group and 30 for the control group. Standard deviations for our implementation were calculated below 1 for all of the cases.

Again for the learned irrelevance case comparisons were made with rabbit data while comparing mean trials to criterion. Once more, there were two phases; in phase 1, some of the subjects are exposed to presentations of uncorrelated data but the others not. In phase 2 they are given presentations of the same stimuli but correlated this time. Rabbits reached the criterion after around 200 for control sit exposure group and amnesic CS/US represented group. However, the control group which is exposed to uncorrelated CS/US pairings reached to the criterion after around 270 trials in the mean. The Gluck and Myers model simulations needed around 80 epochs for control sit

exposure group, around 90 for amnesic sit exposure group, around 80 for amnesic CS/US group. It took around 150 epochs for the control CS/US group. The GeneRec implementation of Gluck and Myers model reached at 212 epochs in mean for control sit exposure group, 79 epochs in mean for amnesic sit exposure group, 47 epochs in mean for amnesic CS/US exposure group and 294 epochs in mean for control CS/US exposure group. The standard deviations were 33, 3, 46 and 142 respectively. The reason for the standard deviation of 142 was due to the occasionally fast reach of model simulations to the criterion, probably caused by formed representations in the hidden layer of cortico-cerebellar network in phase 1.

As a consequence, all of the results obtained were closely correlated with the referred previous findings. However, further representational differentiation and compression cases could be tested on the implemented model to understand its possible uses or even its shortcomings.

5. CONCLUSIONS

5.1 General

Hippocampal-region dysfunction has long been suggested to be an important contributor to the cognitive impairments observed Alzheimer's disease (AD) which is a leading cause of death among people over the age of 60. Recent research has produced findings that may allow early detection of which individuals are most at risk to develop AD in the future. In some elderly individuals, the hippocampus and the endorhinal cortex show signs of atrophy while other nearby brain structures appear intact [1]. It is hoped that study of biologically plausible learning models will probably contribute to our understanding concerning the underlying reasons and possible consequences of hippocampal dysfunction. Therefore the main objective of this thesis work was to develop an artificial neural network model that in many ways behaved like the hippocampal region. For this purpose we have used the cortico-hippocampal model of Gluck and Myers as the basic model. However, we modified this model by the application of the GeneRec algorithm with soft weight bounding and with external inhibition. This model was analyzed through extensive numerical simulations to study phenomena that occur in the hippocampal region, as this region undergoes atrophy.

The results obtained by this work were closely correlated to the results of the original implementation of Gluck and Myers. Most of the time, the GeneRec implementations of the cortico-cerebellar and hippocampal networks learned faster with respect to the original implementations.

As we mentioned before, at a functional level, several neural network models are available for the hippocampal region and its relationship with other brain areas. Although these models may give satisfactory results, considering animal studies, in general their learning algorithms do not care biological plausibility at first. In this work we tried to take into account biological considerations in terms of bounded weights and external inhibition and by using locally available activation variables as opposed to error or other variables. Therefore, we tried to integrate neural network principles with the biological mechanisms to have a better understanding of hippocampal region

dysfunction. Moreover, we have analyzed and compared the results of using the GeneRec algorithm in a different kind of network architecture.

5.2 Recommendations for Future Work

At the beginning of this study our research was concentrated on neural network models of the hippocampal region at a functional level along with learning algorithms that are similar to biological mechanisms. However, we encountered several problems during the GeneRec implementation particularly due to external inhibition. The kWTA inhibition demonstrated its shortcomings such as difficulties to manipulate inhibition under highly dynamic conditions. External inhibition with inhibitory interneurons could not provide the required convergence either. The problem of convergence may have been related to the structure of inhibitory network with its own connections or may be related to the interaction with cortico-hippocampal network. Clearly, the choise of external inhibition method and its application strongly affects the results. Application of alternative inhibition methods or structures may improve convergence with external inhibition and contribute to model's similarity to biological neural networks.

GeneRec was one of the learning algorithm alternatives for this study. Another algorithm may suit better for the purpose of biological plausibility.

Cortico-hippocampal model of Gluck and Myers concentrates on representational differentiation and compression function of hippocampus and its interaction with cortical brain areas. In this study we only studied a few representational differentiation and compression cases hence, study of further cases can improve our understanding of the model and its implementation.

APPENDIX A

TRAINING DATA GENERATOR CODE

```
% cortical/cerebellar & hippocampal network training data set

% - number of network inputs is 18
% - 5 phasic(light, tone, etc.) 13 tonic contextual cues. 5+13=18
% - stimulus patterns are constructed by setting first five bits to 0 or 1,
% depending on the presence or absence of five phasic cues
% - next three bits code for a unique context: 101 context X, 010 context Y
% - the final 10 bits are random string of 0s and 1s, constant across all
% stimulus patterns, but evolve slowly with time, so that on any trial
% there is some probability, P=.01, that one of the 10 bits will be
% inverted; this inversion is permanent unless randomly inverted back.

%-------------------------------------------------
trn_block=[' X-';'AX+';' X-';' Y-';'AY-';' Y-']
% is indeed ['AX+';' X-';'AY-';' Y-']
% No of Presentation of each Stimulus Pattern Matrix being trained
n_pspm=[10,1,10,10,1,10]
trn_set='01a'                % no of TRaiNing SET
%-------------------------------------------------
reply = input('is everything ok up to now? y/n [y]: ','s');
if isempty(reply)
   reply = 'y';
end
if reply~='y'
   clear, return
end
%-------------------------------------------------
cntx_s=round(rand(10,1)); %generates Static CoNTeXt

nib=size(trn_block,1); % No of Items in Block. no of columns
inpt=[];
hpn_out=[];, ccn_out=[];
for n_b=1:nib
   k=trn_block(n_b,:); %read one row at once
   iminpt=inpt; %to add each produced input set to the preceding one
   imhpn_out=hpn_out;, imccn_out=ccn_out;
   if length(k)==3
      if k(1)==' ', indx_p=0; %INDeX of Phasic input. '0':no phasic input
      %index of phasic input 1,2,3,4,5 for A,B,C,D,E respectively.
      elseif k(1)=='A', indx_p=1;
      elseif k(1)=='B', indx_p=2;
      elseif k(1)=='C', indx_p=3;
      elseif k(1)=='D', indx_p=4;
      elseif k(1)=='E', indx_p=5;
```

```
   else disp('improper input case'), return
   end
   if k(2)=='X', cntx_u=[1;0;1]; % Unique CoNTeXt X=[1;0;1]
   elseif k(2)=='Y', cntx_u=[0;1;0]; % Unique CoNTeXt Y=[0;1;0]
   else disp('improper input case'), return
   end
   CRout=k(3); %CRout: Conditioned Response OUTput
   if CRout=='-', CRout=0;
   elseif CRout=='+', CRout=1;
   else disp('improper input case'), return
   end
      else disp('improper input case'), return
      end
      %-----------------------------------------------
      phsc=zeros(5,1);,
      if indx_p~=0
      phsc(indx_p)=1; % PHaSiC input
      end

n_psp=n_pspm(n_b);
inpt=[]; % required, otherwise for the case:
% '# of training data set(i)'<'no of items in training block(nib)' ;
% inpt(:,i) writes on the previous inpt value
hpn_out=[];, ccn_out=[];
for i=1:n_psp % Number of training data sets.
% Since static context should slowly change.
   % Training Input Data. Same for both of the networks
   pinv=100;
   % beta probability distribution function.
   % Flat pdf(probability distrb. func.) with coefficients (1,1)
   rnd=random('beta',1,1)*pinv;
   % with probability of 1/pinv, take inverse of one of the values of
   % static context.
   if round(rnd)==1
      disp('inverse')
      %-----
      sans1=0; % generates integers between 1-10 to choose
      % which one of the static context values will be changed
      while sans1<1 | sans1>10
         sans1=round(random('beta',1,1)*11);
      end
      %-----
      if cntx_s(sans1)==0 %take inverse
      cntx_s(sans1)=1;
      else
      cntx_s(sans1)=0;
      end
   else
```

```
        end
        %input for phasic, unique context X, static context
        inpt(:,i)=[phsc; cntx_u; cntx_s];
        % Training Output Data.
        % Hippocampal Network OUTput. CRout: Conditioned Response OUTput
        hpn_out(:,i)=[inpt(:,i); CRout];
        % Cortical/Cerebellar Network OUTput.
        ccn_out(:,i)=[CRout];
    end
    inpt=[iminpt,inpt];
    hpn_out=[imhpn_out,hpn_out];
    ccn_out=[imccn_out,ccn_out];
end
inpt_cntx=inpt(:,1); hpn_out_cntx=hpn_out(:,1); ccn_out_cntx=ccn_out(:,1);
% Training Data
% inpt, hpn_out, ccn_out
% inpt_cntx, hpn_out_cntx, ccn_out_cntx
% Save training data
fln=['trn_data',trn_set];
save(fln,'trn_block','n_pspm','inpt','hpn_out','ccn_out', ...
    'inpt_cntx','hpn_out_cntx','ccn_out_cntx')
clear
```

APPENDIX B

CORTICO-HIPPOCAMPAL INTACT MODEL CODE

```
% Cort.-Cereb. and Hippocampal Networks (Intact Model). Acquisition Case.
%
% This code simulates intact model for acquisition case.
%
% After network initialization two phases exist in this code:
% 1) Adapt network to context
% 2) Adapt network for acquisition
%
% Algorithm
%
% - Initialize Network (# of layers, neurons, weights, biases)
%
% - Adapt Network
%
%       Minus Phase:
%       1) Clamp only external input to the input units
%       2) Settle the network with the settling routine
%
%       Plus Phase:
%       3) Clamp one training vector (external input and target) to the visible
%      units of the network
%       4) Settle the network with the settling routine
%          (only changes activations of units, not the weights), therefore get
%      the activations
%
%       Weight Change:
%       5) Calculate and apply appropriate weight changes
%          perform "Soft Weight Bounding"
%          calculate individual mean square errors
%       6) Repeat steps 1 through 5 for all training vectors (1 epoch)
%          calculate overall mean square errors for one epoch
%       7) Repeat steps 1 through 6 until the mean square error is below a
%      threshold
%
%   Settling Routine(synchronous updating):
%   1) Force the outputs of all visible units to the specified input vector
%   2) Assign "0" output value to all unknown units
%   3) For all units: compute raw_netin = sum sending_act * weight
%      (Add both top-down input dot products and bottom-up input dot products)
%   4) For all units compute:
%
%       netin = netin + dt * (raw_netin - netin)
%       act = sigmoid(netin)
%
```

```
%   5) Repeat steps 3 to 4 for several processing cycles, until activation
%      changes go below a threshold

% Ilim Cagiran, 2006
% Revision: 17_c

clear
tic

load trn_data02.mat

nmax_loop_settle= 100;  % Max Number of Loops for hpn Settling
z_ninL2_hpn= 0.9;
%----------------------

% CREATE NETWORK DATA
%-----------------------------------------------
Si= 18; % Size of Input vector
S1_hpn= 10; % Size of Hidden Layer of HPN
S2_hpn= 19; % Size of Output Layer of HPN
S1_ccn= 60; % Size of Hidden Layer of CCN
S2_ccn= 1; % Size of Output Layer of CCN

% ----------------------
% Due to "soft weight bounding" weight range is (0,1).
% Therefore weight values are centered on the middle value of 0.5 instead of 0

% Intialize HPN
U_hpn= 0.1;
% All weights of the net initialized according to random distribution U
% WEIGHTS: The indices in "W_J_K" designates the weight matrix from
% the J'th layer to the K'th layer in this code.
% Input layer is the 1st layer. Hidden layer is the 2nd layer.
% Output layer is the 3rd layer.
% rands: returns an S-by-R weight matrix of random values between -1 and 1
W_hpn_1_2= U_hpn*rands(S1_hpn,Si)+0.5;
W_hpn_2_1= W_hpn_1_2'; % take transpose of weight matrix.
% Derivation of the algorithm requires symmetric weights.
W_hpn_2_3= U_hpn*rands(S2_hpn,S1_hpn)+0.5;
W_hpn_3_2= W_hpn_2_3';
% Biases
b_hpn_2= zeros(S1_hpn,1); % Leabra default 0 with 0 variance
b_hpn_3= zeros(S2_hpn,1);

% Intialize CCN
U_ccn_lo= 0.3; %U LOwer. Due to SWB, max value of U can only be 0.5
U_ccn_up= 0.1; %U UPper.
% "W_ccn_2e_3i" designates the weight matrix from the 2nd Excitatory layer
```

```
% to the 3th Inhibitory layer in this code.
% W_ccn_1e_2e= U_ccn_lo*rands(S1_ccn,Si)+0.5;
% W_ccn_1e_2i= U_ccn_lo*rands(S1_ccn_i,Si)+0.5;
W_ccn_2e_3e= U_ccn_up*rands(S2_ccn,S1_ccn)+0.5;
W_ccn_3e_2e= W_ccn_2e_3e';

% Biases
b_ccn_2= zeros(S1_ccn,1);
b_ccn_3= zeros(S2_ccn,1);

% dt. temporal integration constant for Excitatory neurons. Step Constant
sc_e= 0.3;
% dt. temporal integration constant for Inhibitory neurons
sc_i= 0.4;
% overall weight scale for FeedForward inhibition. multiplied with weights
% for different layer level excitatory to inhibitory connections.
scale_ff= 1 ;
% overall weight scale for FeedBack inhibition. multiplied with weights
% for same layer level excitatory to inhibitory connections.
scale_fb= 1 ;
scale_ff_up= 1 ;
scale_fb_up= 1 ;
w_mult_3i_3e= 1 ; % weight multiplier for 3i to 3e connection.
w_mult_2e_3e= 1 ;
w_mult_2e_3i= 1 ;
% w_mult_1e_2e= 1/8 ;
% w_mult_1e_2i= 1/8 ;
w_mult_2e_2i= 1 ;
lr_mult_ccn= 1; % Learning Rate Multiplier ccn
% ----------------------

% Intialize HPN CCN connection
Uvhc= 1;
Vhc= Uvhc*rands(S1_ccn, S1_hpn)+0.0; % hpn-ccn connection weights.
w_mult_h2e_c2e= 1 ;
% ----------------------

% ADAPT NETWORK TO CONTEXT
%-------------------------------------------------
n_ep= 300;
mse_log=zeros(n_ep,5); % to improve program performance
disp('Please wait')
q= 0.25; % 0 < q < 1 where q = 0.25 may be default
% weight change threshold. default is 0.1 in GeneRec
dwt_thresh_ccn= 1.e-3; dwt_thresh_hpn= 1.e-2;
[r,c]=size(inpt_cntx);
ki2=50; % after how many epochs results will be showed
ki2=ceil(n_ep/(ki2*30))*ki2;
```

```
for ko2=1:n_ep
   mse_o_hpn = 0; mse_o_ccn = 0; mse_thr_o_hpn = 0; mse_thr_o_ccn = 0;
   % mean values are required for bar graphs
   m_o_aL3_ccn_m = 0;  m_o_aL2_ccn_m = 0;
   for cnt2=1:c
      aL1_hpn= inpt_cntx(:,cnt2); y_hpn= hpn_out_cntx(:,cnt2);
      aL1_ccn= inpt_cntx(:,cnt2); y_ccn= ccn_out_cntx(:,cnt2);
      if y_hpn(end)==1    % conditional response target is 1
         lr_hpn= 0.25; % Learning Rate for hpn
         lr_ccn_lo= lr_mult_ccn*0.1; % Learning Rate LOwer layer
         lr_ccn_up= lr_mult_ccn*0.5; % Learning Rate UPper layer
      elseif y_hpn(end)==0
         lr_hpn= 0.025;
         lr_ccn_lo= lr_mult_ccn*0.01;
         lr_ccn_up= lr_mult_ccn*0.05;
      else
         error('Conditional Response Target should be either 0 or 1')
      end

      % MINUS PHASE
      %-----------------------

      % HPN OUTPUT CALCULATION / STARTS - MINUS PHASE
      %---------------------------------------------
      % Assign "0" output value to all unknown units
      aL2_hpn= zeros(S1_hpn,1);
      aL3_hpn= zeros(S2_hpn,1);

      ninL2_hpn= zeros(S1_hpn,1); % NetIN, initial values
      ninL3_hpn= zeros(S2_hpn,1);
      sc= 0.1; % Step Constant, dt

      warning off MATLAB:divideByZero
      delta_aL2_hpn=1; delta_aL3_hpn=1;
      n_loop_hpn_m= 0; % Number of Loops for hpn settling in Minus phase
      for cnt5=1:nmax_loop_settle
         % _pr means Previous. used to calculate change
         aL2_hpn_pr= aL2_hpn;
         aL3_hpn_pr= aL3_hpn;

         % Add both bottom-up & top-down input dot products
         % Layer2
         nL2_hpn= W_hpn_1_2*aL1_hpn + W_hpn_3_2*aL3_hpn + b_hpn_2;
         % Layer3. output units are not self connected. No top-down input
         nL3_hpn= W_hpn_2_3*aL2_hpn + b_hpn_3;

         % netin = netin + dt * (raw_netin - netin)
         ninL2_hpn= ninL2_hpn + sc*(nL2_hpn - ninL2_hpn);
```

```
    ninL3_hpn= ninL3_hpn + sc*(nL3_hpn - ninL3_hpn);

    % act = sigmoid(netin+inhibition)
    aL2_hpn= 1 ./ (1 + exp(-ninL2_hpn)); % logsigmoid
    aL3_hpn= 1 ./ (1 + exp(-ninL3_hpn)); % logsigmoid

    % average of activations change ratio
    delta_aL2_hpn= mean(abs((aL2_hpn-aL2_hpn_pr)./aL2_hpn_pr));
    delta_aL3_hpn= mean(abs((aL3_hpn-aL3_hpn_pr)./aL3_hpn_pr));

    if (delta_aL2_hpn < dwt_thresh_hpn) & ...
          (delta_aL3_hpn < dwt_thresh_hpn), break, end
end
aL2_hpn_m= aL2_hpn; % _m, Minus
aL3_hpn_m= aL3_hpn;

% CCN OUTPUT CALCULATION / STARTS - MINUS PHASE
%----------------------------------------------
% Assign "0" output value to all unknown units
aL2_ccn= zeros(S1_ccn,1);
aL3_ccn= zeros(S2_ccn,1);

ninL2_ccn= zeros(S1_ccn,1); % NetIN, initial values
ninL3_ccn= zeros(S2_ccn,1);

warning off MATLAB:divideByZero
delta_aL2_ccn=1; delta_aL3_ccn=1;
while (delta_aL2_ccn > dwt_thresh_ccn) | ...
      (delta_aL3_ccn > dwt_thresh_ccn)
    % _pr means Previous. used to calculate change.
    aL2_ccn_pr= aL2_ccn; aL3_ccn_pr= aL3_ccn;

    % Add both bottom-up & top-down input dot products.
    % For inhibitory connections use "-".
    nL2_ccn= w_mult_h2e_c2e*(Vhc*aL2_hpn_m)    ...
      + W_ccn_3e_2e*aL3_ccn + b_ccn_2; % Layer 2e
    % Layer3e. output units are not self connected. No top-down input
    nL3_ccn= w_mult_2e_3e*W_ccn_2e_3e*aL2_ccn + b_ccn_3;

    % netin = netin + dt * (raw_netin - netin)
    ninL2_ccn= ninL2_ccn + sc_e*(nL2_ccn - ninL2_ccn);
    ninL3_ccn= ninL3_ccn + sc_e*(nL3_ccn - ninL3_ccn);

    aL2_ccn= 1 ./ (1 + exp(-ninL2_ccn)); % logsigmoid
    aL3_ccn= 1 ./ (1 + exp(-ninL3_ccn)); % logsigmoid

    % average of activations change ratio
    delta_aL2_ccn= mean(abs((aL2_ccn-aL2_ccn_pr)./aL2_ccn_pr));
```

```
    delta_aL3_ccn= mean(abs((aL3_ccn-aL3_ccn_pr)./aL3_ccn_pr));

end
aL2_ccn_m= aL2_ccn; aL3_ccn_m= aL3_ccn; % _m, Minus
if y_hpn(end)==1    % conditional response target is 1
    cr_ccn(ko2) = aL3_ccn_m; % Conditional Response ccn
end

% PLUS PHASE
%-----------------------

% HPN OUTPUT CALCULATION / STARTS - PLUS PHASE
%--------------------------------------------
% Assign "0" output value to all unknown units
aL2_hpn= zeros(S1_hpn,1);
aL3_hpn= y_hpn;

ninL2_hpn= zeros(S1_hpn,1); % NetIN, initial values
sc= 0.1; % Step Constant, dt

warning off MATLAB:divideByZero
delta_aL2_hpn=1;
n_loop_hpn_p= 0; % Number of Loops for hpn settling in Minus phase
for cnt5=1:nmax_loop_settle
    % _pr means Previous. used to calculate change
    aL2_hpn_pr= aL2_hpn;

    % Add both bottom-up & top-down input dot products
    % Layer2
    nL2_hpn= W_hpn_1_2*aL1_hpn + W_hpn_3_2*aL3_hpn + b_hpn_2;

    % netin = netin + dt * (raw_netin - netin)
    ninL2_hpn= ninL2_hpn + sc*(nL2_hpn - ninL2_hpn);

    % act = sigmoid(netin+inhibition)
    aL2_hpn= 1 ./ (1 + exp(-ninL2_hpn)); % logsigmoid

    % average of activations change ratio
    delta_aL2_hpn= mean(abs((aL2_hpn-aL2_hpn_pr)./aL2_hpn_pr));

    if (delta_aL2_hpn < dwt_thresh_hpn), break, end
end
aL2_hpn_p= aL2_hpn; aL3_hpn_p= y_hpn; % _p, Plus

% CCN OUTPUT CALCULATION / STARTS - PLUS PHASE
%--------------------------------------------
% Assign "0" output value to all unknown units
aL2_ccn= zeros(S1_ccn,1);
```

```
aL3_ccn= y_ccn;

ninL2_ccn= zeros(S1_ccn,1); % NetIN, initial values

warning off MATLAB:divideByZero
delta_aL2_ccn=1;
while delta_aL2_ccn > dwt_thresh_ccn
   % _pr means Previous. used to calculate change
   aL2_ccn_pr= aL2_ccn;

   % Add both bottom-up & top-down input dot products.
   % For inhibitory connections use "-".
   nL2_ccn= w_mult_h2e_c2e*(Vhc*aL2_hpn_p)    ...
      + W_ccn_3e_2e*aL3_ccn + b_ccn_2; % Layer 2e

   % netin = netin + dt * (raw_netin - netin)
   % act = sigmoid(netin)
   ninL2_ccn= ninL2_ccn + sc_e*(nL2_ccn - ninL2_ccn);

   aL2_ccn= 1 ./ (1 + exp(-ninL2_ccn)); % logsigmoid

   % average of activations change ratio
   delta_aL2_ccn= mean(abs((aL2_ccn-aL2_ccn_pr)./aL2_ccn_pr));

end
aL2_ccn_p= aL2_ccn; aL3_ccn_p= y_ccn; % _p, Plus


% CALCULATE WEIGHT CHANGES
%-------------------------
% delta_W_J_K= learning rate*(x_J_plus*y_K_plus - x_J_minus*y_K_minus)
% sending unit with activation x_J
% receiving unit with activation y_K

% in "minus phase" aL1 is clamped
% in "plus phase" aL1 and aL3 are both clamped

% HPN WEIGHT UPDATE
%----------------------------------------
W_ch_hpn_1_2=zeros(S1_hpn,Si);
for cnt3=1:Si
   W_ch_hpn_1_2(:,cnt3)= lr_hpn* ( aL1_hpn(cnt3)*aL2_hpn_p ...
      - aL1_hpn(cnt3)*aL2_hpn_m );
end
% weight update with SWB W change
W_hpn_1_2= W_hpn_1_2 + W_ch_hpn_1_2;
% --------------
W_ch_hpn_2_1=zeros(Si,S1_hpn);
```

```
for cnt3=1:S1_hpn
   W_ch_hpn_2_1(:,cnt3)= lr_hpn* ( aL2_hpn_p(cnt3)*aL1_hpn ...
      - aL2_hpn_m(cnt3)*aL1_hpn );
end
% weight update with SWB W change
W_hpn_2_1= W_hpn_2_1 + W_ch_hpn_2_1;
% --------------
W_ch_hpn_2_3=zeros(S2_hpn,S1_hpn);
for cnt3=1:S1_hpn
   W_ch_hpn_2_3(:,cnt3)= lr_hpn* ( aL2_hpn_p(cnt3)*aL3_hpn_p ...
      - aL2_hpn_m(cnt3)*aL3_hpn_m );
end
% weight update with SWB W change
W_hpn_2_3= W_hpn_2_3 + W_ch_hpn_2_3;
% --------------
W_ch_hpn_3_2=zeros(S1_hpn,S2_hpn);
for cnt3=1:S2_hpn
   W_ch_hpn_3_2(:,cnt3)= lr_hpn* ( aL3_hpn_p(cnt3)*aL2_hpn_p ...
      - aL3_hpn_m(cnt3)*aL2_hpn_m );
end
% weight update with SWB W change
W_hpn_3_2= W_hpn_3_2 + W_ch_hpn_3_2;
% --------------
% HPN BIAS UPDATE
b_hpn_2= b_hpn_2 + lr_hpn*(aL2_hpn_p-aL2_hpn_m);
b_hpn_3= b_hpn_3 + lr_hpn*(aL3_hpn_p-aL3_hpn_m);

% CCN WEIGHT UPDATE
%---------------------------------------
W_ch_ccn_2e_3e=zeros(S2_ccn,S1_ccn);
for cnt3=1:S1_ccn
   W_ch_ccn_2e_3e(:,cnt3)= lr_ccn_up*( aL2_ccn_p(cnt3)*aL3_ccn_p ...
      - aL2_ccn_m(cnt3)*aL3_ccn_m );
end
W_ccn_2e_3e= W_ccn_2e_3e + W_ch_ccn_2e_3e;
% --------------
W_ch_ccn_3e_2e=zeros(S1_ccn,S2_ccn);
for cnt3=1:S2_ccn
   W_ch_ccn_3e_2e(:,cnt3)= lr_ccn_up*( aL3_ccn_p(cnt3)*aL2_ccn_p ...
      - aL3_ccn_m(cnt3)*aL2_ccn_m );
end
W_ccn_3e_2e= W_ccn_3e_2e + W_ch_ccn_3e_2e;
% --------------
% CCN BIAS UPDATE
b_ccn_2= b_ccn_2 + lr_ccn_lo*(aL2_ccn_p-aL2_ccn_m);
b_ccn_3= b_ccn_3 + lr_ccn_up*(aL3_ccn_p-aL3_ccn_m);
```

```
    % CALCULATE MS ERRORS & DATA REQUIRED
    %---------------------------------------
    % HPN ERRORS
    % mean square error individual
    mse_i_hpn= sum( ( y_hpn - aL3_hpn_m ).^2 )/S2_hpn;
    % mean square error individual. thresholded, x=0 if x<.5 & x=1 if x>=.5
    mse_thr_i_hpn= sum( ( y_hpn - round(aL3_hpn_m) ).^2 )/S2_hpn;
    %-----------------------
    mse_o_hpn = mse_o_hpn + mse_i_hpn; % mse overall for one epoch
    % mse_thr overall for one epoch
    mse_thr_o_hpn = mse_thr_o_hpn + mse_thr_i_hpn;

    % CCN ERRORS
    % mean square error individual
    mse_i_ccn= sum( ( y_ccn - aL3_ccn_m ).^2 )/S2_ccn;
    % mean square error individual. thresholded, x=0 if x<.5 & x=1 if x>=.5
    mse_thr_i_ccn= sum( ( y_ccn - round(aL3_ccn_m) ).^2 )/S2_ccn;
    %-----------------------
    mse_o_ccn = mse_o_ccn + mse_i_ccn; % mse overall for one epoch
    % mse_thr overall for one epoch
    mse_thr_o_ccn = mse_thr_o_ccn + mse_thr_i_ccn;
    %-----------------------
    % mean values are required for bar graphs
    m_o_aL3_ccn_m = m_o_aL3_ccn_m + aL3_ccn_m;
    m_o_aL2_ccn_m = m_o_aL2_ccn_m + aL2_ccn_m;

end

% mse log data: 1st column is # of epoch, 2nd is mse_o_hpn,
% 3rd is mse_o_ccn, 4th is mse_thr_o_hpn, 5th is mse_thr_o_ccn
mse_o_hpn = mse_o_hpn/c; mse_o_ccn = mse_o_ccn/c;
mse_thr_o_hpn= mse_thr_o_hpn/c; mse_thr_o_ccn= mse_thr_o_ccn/c;
mse_log(ko2,1)=ko2; mse_log(ko2,2)=mse_o_hpn; mse_log(ko2,3)=mse_o_ccn;
mse_log(ko2,4)=mse_thr_o_hpn; mse_log(ko2,5)=mse_thr_o_ccn;
% mean values are required for bar graphs
m_o_aL3_ccn_m = m_o_aL3_ccn_m/c;  m_o_aL2_ccn_m = m_o_aL2_ccn_m/c;

if rem(ko2,ki2)==0 | ko2==1 | ko2==n_ep

    dtxt3=num2str(mse_o_hpn);
    dtxt3b=num2str(mse_thr_o_hpn);
    dtxt4=num2str(ko2);
    %---------------
    dtxt5=num2str(mse_o_ccn);
    dtxt5b=num2str(mse_thr_o_ccn);
    dtxtg7=['mse_ccn=', dtxt5,'  &  mse_thr_ccn=', dtxt5b,...
            '  at epoch ', dtxt4];
    disp(dtxtg7)
```

```
   end

   subplot(2,3,2), bar(m_o_aL3_ccn_m), axis([0,2,0,1]), ...
      TITLE('Output Layer Activity ')
   subplot(2,3,5), bar(m_o_aL2_ccn_m), axis([0,61,0,1]), ...
      TITLE('Hidden Layer Activity ')

   M(ko2) = getframe(gcf);
end
toc

% Plot Performance of Network
figure
semilogy(mse_log(:,1), mse_log(:,2), 'b--',mse_log(:,1),mse_log(:,3),'g--',...
        mse_log(:,1), mse_log(:,4), 'b-', mse_log(:,1),mse_log(:,5),'g-')
TITLE('PERFORMANCE PLOT, GENEREC, SWB')
XLABEL('EPOCHS')
YLABEL('MSE, blue:hpn, green:ccn, solid:thresholded, dashed:raw')

% ========================================================
figure

% ADAPT NETWORK
%-----------------------------------------------
n_ep= 500;
mse_log=zeros(n_ep,5); % to improve program performance
disp('Please wait')
q= 0.25; % 0 < q < 1 where q = 0.25 may be default
% weight change threshold. default is 0.1 in GeneRec
dwt_thresh_ccn= 1.e-3; dwt_thresh_hpn= 1.e-2;
[r,c]=size(inpt);
ki2=50; % after how many epochs results will be showed
ki2=ceil(n_ep/(ki2*30))*ki2;
for ko2=1:n_ep
   mse_o_hpn = 0; mse_o_ccn = 0; mse_thr_o_hpn = 0; mse_thr_o_ccn = 0;
   % mean values are required for bar graphs
   m_o_aL3_ccn_m = 0;  m_o_aL2_ccn_m = 0;
   for cnt2=1:c
      aL1_hpn= inpt(:,cnt2); y_hpn= hpn_out(:,cnt2);
      aL1_ccn= inpt(:,cnt2); y_ccn= ccn_out(:,cnt2);
      if y_hpn(end)==1    % conditional response target is 1
         lr_hpn= 0.25;   % Learning Rate for hpn
         lr_ccn_lo= lr_mult_ccn*0.1; % Learning Rate LOwer layer
         lr_ccn_up= lr_mult_ccn*0.5; % Learning Rate UPper layer
      elseif y_hpn(end)==0
         lr_hpn= 0.025;
         lr_ccn_lo= lr_mult_ccn*0.01;
         lr_ccn_up= lr_mult_ccn*0.05;
```

```
else
   error('Conditional Response Target should be either 0 or 1')
end

% MINUS PHASE
%-----------------------

% HPN OUTPUT CALCULATION / STARTS - MINUS PHASE
%---------------------------------------------
% Assign "0" output value to all unknown units
aL2_hpn= zeros(S1_hpn,1);
aL3_hpn= zeros(S2_hpn,1);

ninL2_hpn= zeros(S1_hpn,1); % NetIN, initial values
ninL3_hpn= zeros(S2_hpn,1);
sc= 0.1; % Step Constant, dt

warning off MATLAB:divideByZero
delta_aL2_hpn=1; delta_aL3_hpn=1;
n_loop_hpn_m= 0; % Number of Loops for hpn settling in Minus phase
for cnt5=1:nmax_loop_settle
   % _pr means Previous. used to calculate change
   aL2_hpn_pr= aL2_hpn;
   aL3_hpn_pr= aL3_hpn;

   % Add both bottom-up & top-down input dot products
   % Layer2
   nL2_hpn= W_hpn_1_2*aL1_hpn + W_hpn_3_2*aL3_hpn + b_hpn_2;
   % Layer3. output units are not self connected. No top-down input
   nL3_hpn= W_hpn_2_3*aL2_hpn + b_hpn_3;

   % netin = netin + dt * (raw_netin - netin)
   ninL2_hpn= ninL2_hpn + sc*(nL2_hpn - ninL2_hpn);
   ninL3_hpn= ninL3_hpn + sc*(nL3_hpn - ninL3_hpn);

   % act = sigmoid(netin+inhibition)
   aL2_hpn= 1 ./ (1 + exp(-ninL2_hpn)); % logsigmoid
   aL3_hpn= 1 ./ (1 + exp(-ninL3_hpn)); % logsigmoid

   % average of activations change ratio
   delta_aL2_hpn= mean(abs((aL2_hpn-aL2_hpn_pr)./aL2_hpn_pr));
   delta_aL3_hpn= mean(abs((aL3_hpn-aL3_hpn_pr)./aL3_hpn_pr));

   if (delta_aL2_hpn < dwt_thresh_hpn) & ...
         (delta_aL3_hpn < dwt_thresh_hpn), break, end
end
aL2_hpn_m= aL2_hpn; % _m, Minus
aL3_hpn_m= aL3_hpn;
```

```
% CCN OUTPUT CALCULATION / STARTS - MINUS PHASE
%-----------------------------------------------
% Assign "0" output value to all unknown units
aL2_ccn= zeros(S1_ccn,1);
aL3_ccn= zeros(S2_ccn,1);

ninL2_ccn= zeros(S1_ccn,1); % NetIN, initial values
ninL3_ccn= zeros(S2_ccn,1);

warning off MATLAB:divideByZero
delta_aL2_ccn=1; delta_aL3_ccn=1;
while (delta_aL2_ccn > dwt_thresh_ccn) | ...
      (delta_aL3_ccn > dwt_thresh_ccn)
   % _pr means Previous. used to calculate change.
   aL2_ccn_pr= aL2_ccn; aL3_ccn_pr= aL3_ccn;

   % Add both bottom-up & top-down input dot products.
   % For inhibitory connections use "-".
   nL2_ccn= w_mult_h2e_c2e*(Vhc*aL2_hpn_m)    ...
      + W_ccn_3e_2e*aL3_ccn + b_ccn_2; % Layer 2e
   % Layer3e. output units are not self connected. No top-down input
   nL3_ccn= w_mult_2e_3e*W_ccn_2e_3e*aL2_ccn + b_ccn_3;

   % netin = netin + dt * (raw_netin - netin)
   ninL2_ccn= ninL2_ccn + sc_e*(nL2_ccn - ninL2_ccn);
   ninL3_ccn= ninL3_ccn + sc_e*(nL3_ccn - ninL3_ccn);

   aL2_ccn= 1 ./ (1 + exp(-ninL2_ccn)); % logsigmoid
   aL3_ccn= 1 ./ (1 + exp(-ninL3_ccn)); % logsigmoid

   % average of activations change ratio
   delta_aL2_ccn= mean(abs((aL2_ccn-aL2_ccn_pr)./aL2_ccn_pr));
   delta_aL3_ccn= mean(abs((aL3_ccn-aL3_ccn_pr)./aL3_ccn_pr));

end
aL2_ccn_m= aL2_ccn; aL3_ccn_m= aL3_ccn; % _m, Minus
if y_hpn(end)==1    % conditional response target is 1
   cr_ccn(ko2) = aL3_ccn_m; % Conditional Response ccn
end

% PLUS PHASE
%-----------------------

% HPN OUTPUT CALCULATION / STARTS - PLUS PHASE
%-----------------------------------------------
% Assign "0" output value to all unknown units
aL2_hpn= zeros(S1_hpn,1);
```

```
aL3_hpn= y_hpn;

ninL2_hpn= zeros(S1_hpn,1); % NetIN, initial values
sc= 0.1; % Step Constant, dt

warning off MATLAB:divideByZero
delta_aL2_hpn=1;
n_loop_hpn_p= 0; % Number of Loops for hpn settling in Minus phase
for cnt5=1:nmax_loop_settle
   % _pr means Previous. used to calculate change
   aL2_hpn_pr= aL2_hpn;

   % Add both bottom-up & top-down input dot products
   % Layer2
   nL2_hpn= W_hpn_1_2*aL1_hpn + W_hpn_3_2*aL3_hpn + b_hpn_2;

   % netin = netin + dt * (raw_netin - netin)
   ninL2_hpn= ninL2_hpn + sc*(nL2_hpn - ninL2_hpn);

   % act = sigmoid(netin+inhibition)
   aL2_hpn= 1 ./ (1 + exp(-ninL2_hpn)); % logsigmoid

   % average of activations change ratio
   delta_aL2_hpn= mean(abs((aL2_hpn-aL2_hpn_pr)./aL2_hpn_pr));

   if (delta_aL2_hpn < dwt_thresh_hpn), break, end
end
aL2_hpn_p= aL2_hpn; aL3_hpn_p= y_hpn; % _p, Plus

% CCN OUTPUT CALCULATION / STARTS - PLUS PHASE
%---------------------------------------------
% Assign "0" output value to all unknown units
aL2_ccn= zeros(S1_ccn,1);
aL3_ccn= y_ccn;

ninL2_ccn= zeros(S1_ccn,1); % NetIN, initial values

warning off MATLAB:divideByZero
delta_aL2_ccn=1;
while delta_aL2_ccn > dwt_thresh_ccn
   % _pr means Previous. used to calculate change
   aL2_ccn_pr= aL2_ccn;

   % Add both bottom-up & top-down input dot products.
   % For inhibitory connections use "-".
   nL2_ccn= w_mult_h2e_c2e*(Vhc*aL2_hpn_p)    ...
      + W_ccn_3e_2e*aL3_ccn + b_ccn_2; % Layer 2e
```

```
    % netin = netin + dt * (raw_netin - netin)
    % act = sigmoid(netin)
    ninL2_ccn= ninL2_ccn + sc_e*(nL2_ccn - ninL2_ccn);

    aL2_ccn= 1 ./ (1 + exp(-ninL2_ccn)); % logsigmoid

    % average of activations change ratio
    delta_aL2_ccn= mean(abs((aL2_ccn-aL2_ccn_pr)./aL2_ccn_pr));

end
aL2_ccn_p= aL2_ccn; aL3_ccn_p= y_ccn; % _p, Plus


% CALCULATE WEIGHT CHANGES
%-------------------------
% delta_W_J_K= learning rate*(x_J_plus*y_K_plus - x_J_minus*y_K_minus)
% sending unit with activation x_J
% receiving unit with activation y_K

% in "minus phase" aL1 is clamped
% in "plus phase" aL1 and aL3 are both clamped

% HPN WEIGHT UPDATE
%---------------------------------------
W_ch_hpn_1_2=zeros(S1_hpn,Si);
for cnt3=1:Si
    W_ch_hpn_1_2(:,cnt3)= lr_hpn* ( aL1_hpn(cnt3)*aL2_hpn_p ...
        - aL1_hpn(cnt3)*aL2_hpn_m );
end
% weight update with SWB W change
W_hpn_1_2= W_hpn_1_2 + W_ch_hpn_1_2;
% --------------
W_ch_hpn_2_1=zeros(Si,S1_hpn);
for cnt3=1:S1_hpn
    W_ch_hpn_2_1(:,cnt3)= lr_hpn* ( aL2_hpn_p(cnt3)*aL1_hpn ...
        - aL2_hpn_m(cnt3)*aL1_hpn );
end
% weight update with SWB W change
W_hpn_2_1= W_hpn_2_1 + W_ch_hpn_2_1;
% --------------
W_ch_hpn_2_3=zeros(S2_hpn,S1_hpn);
for cnt3=1:S1_hpn
    W_ch_hpn_2_3(:,cnt3)= lr_hpn* ( aL2_hpn_p(cnt3)*aL3_hpn_p ...
        - aL2_hpn_m(cnt3)*aL3_hpn_m );
end
% weight update with SWB W change
W_hpn_2_3= W_hpn_2_3 + W_ch_hpn_2_3;
% --------------
```

```
W_ch_hpn_3_2=zeros(S1_hpn,S2_hpn);
for cnt3=1:S2_hpn
   W_ch_hpn_3_2(:,cnt3)= lr_hpn* ( aL3_hpn_p(cnt3)*aL2_hpn_p ...
      - aL3_hpn_m(cnt3)*aL2_hpn_m );
end
% weight update with SWB W change
W_hpn_3_2= W_hpn_3_2 + W_ch_hpn_3_2;
% --------------
% HPN BIAS UPDATE
b_hpn_2= b_hpn_2 + lr_hpn*(aL2_hpn_p-aL2_hpn_m);
b_hpn_3= b_hpn_3 + lr_hpn*(aL3_hpn_p-aL3_hpn_m);

% CCN WEIGHT UPDATE
%---------------------------------------
W_ch_ccn_2e_3e=zeros(S2_ccn,S1_ccn);
for cnt3=1:S1_ccn
   W_ch_ccn_2e_3e(:,cnt3)= lr_ccn_up*( aL2_ccn_p(cnt3)*aL3_ccn_p ...
      - aL2_ccn_m(cnt3)*aL3_ccn_m );
end
W_ccn_2e_3e= W_ccn_2e_3e + W_ch_ccn_2e_3e ;
% --------------
W_ch_ccn_3e_2e=zeros(S1_ccn,S2_ccn);
for cnt3=1:S2_ccn
   W_ch_ccn_3e_2e(:,cnt3)= lr_ccn_up*( aL3_ccn_p(cnt3)*aL2_ccn_p ...
      - aL3_ccn_m(cnt3)*aL2_ccn_m );
end
W_ccn_3e_2e= W_ccn_3e_2e + W_ch_ccn_3e_2e ;
% --------------
% CCN BIAS UPDATE
b_ccn_2= b_ccn_2 + lr_ccn_lo*(aL2_ccn_p-aL2_ccn_m);
b_ccn_3= b_ccn_3 + lr_ccn_up*(aL3_ccn_p-aL3_ccn_m);


% CALCULATE MS ERRORS & DATA REQUIRED
%---------------------------------------
% HPN ERRORS
% mean square error individual
mse_i_hpn= sum( ( y_hpn - aL3_hpn_m ).^2 )/S2_hpn;
% mean square error individual. thresholded, x=0 if x<.5 & x=1 if x>=.5
mse_thr_i_hpn= sum( ( y_hpn - round(aL3_hpn_m) ).^2 )/S2_hpn;
%-----------------------
mse_o_hpn = mse_o_hpn + mse_i_hpn; % mse overall for one epoch
% mse_thr overall for one epoch
mse_thr_o_hpn = mse_thr_o_hpn + mse_thr_i_hpn;

% CCN ERRORS
% mean square error individual
mse_i_ccn= sum( ( y_ccn - aL3_ccn_m ).^2 )/S2_ccn;
```

```
   % mean square error individual. thresholded, x=0 if x<.5 & x=1 if x>=.5
   mse_thr_i_ccn= sum( ( y_ccn - round(aL3_ccn_m) ).^2 )/S2_ccn;
   %-----------------------
   mse_o_ccn = mse_o_ccn + mse_i_ccn; % mse overall for one epoch
   % mse_thr overall for one epoch
   mse_thr_o_ccn = mse_thr_o_ccn + mse_thr_i_ccn;

   %-----------------------
   % mean values are required for bar graphs
   m_o_aL3_ccn_m = m_o_aL3_ccn_m + aL3_ccn_m;
   m_o_aL2_ccn_m = m_o_aL2_ccn_m + aL2_ccn_m;

end

% mse log data: 1st column is # of epoch, 2nd is mse_o_hpn,
% 3rd is mse_o_ccn, 4th is mse_thr_o_hpn, 5th is mse_thr_o_ccn
mse_o_hpn = mse_o_hpn/c; mse_o_ccn = mse_o_ccn/c;
mse_thr_o_hpn= mse_thr_o_hpn/c; mse_thr_o_ccn= mse_thr_o_ccn/c;
mse_log(ko2,1)=ko2; mse_log(ko2,2)=mse_o_hpn; mse_log(ko2,3)=mse_o_ccn;
mse_log(ko2,4)=mse_thr_o_hpn; mse_log(ko2,5)=mse_thr_o_ccn;
% mean values are required for bar graphs
m_o_aL3_ccn_m = m_o_aL3_ccn_m/c;  m_o_aL2_ccn_m = m_o_aL2_ccn_m/c;
% mean ccn output log: 1st column is # of epoch, 2nd is cr_ccn,
% 3rd is m_o_aL3_ccn_m,
% 4th is difference(should be positive): cr_ccn - m_o_ccn_out
m_o_ccn_out(ko2,1)=ko2; m_o_ccn_out(ko2,2)=cr_ccn(ko2);
m_o_ccn_out(ko2,3)=m_o_aL3_ccn_m;
m_o_ccn_out(ko2,4)=cr_ccn(ko2)-m_o_aL3_ccn_m;

if rem(ko2,ki2)==0 | ko2==1 | ko2==n_ep

   dtxt3=num2str(mse_o_hpn);
   dtxt3b=num2str(mse_thr_o_hpn);
   dtxt4=num2str(ko2);
   %---------------
   dtxt5=num2str(mse_o_ccn);
   dtxt5b=num2str(mse_thr_o_ccn);
   dtxtg7=['mse_ccn=', dtxt5,'  &  mse_thr_ccn=', dtxt5b,...
         '  at epoch ', dtxt4];
   disp(dtxtg7)
end
subplot(2,3,1), bar(cr_ccn(ko2)), axis([0,2,0,1]), ...
   TITLE('CR CCN Activity ')
subplot(2,3,2), bar(m_o_aL3_ccn_m), axis([0,2,0,1]), ...
   TITLE('Output Layer Activity ')
subplot(2,3,5), bar(m_o_aL2_ccn_m), axis([0,61,0,1]), ...
   TITLE('Hidden Layer Activity ')
```

```
   M(ko2) = getframe(gcf);
end
toc

% Plot Performance of Network
figure
semilogy(mse_log(:,1), mse_log(:,2), 'b--',mse_log(:,1),mse_log(:,3),'g--',...
        mse_log(:,1), mse_log(:,4), 'b-', mse_log(:,1),mse_log(:,5),'g-')
TITLE('PERFORMANCE PLOT, GENEREC, SWB')
XLABEL('EPOCHS')
YLABEL('MSE, blue:hpn, green:ccn, solid:thresholded, dashed:raw')

disp('mse_log & ccn_out log will be saved as gr_ccnhpn_????.mat ')
reply = input('Enter the last part of the file name (eg. trial2):  ','s');
fln=['gr_ccnhpn_', reply];
save(fln, 'mse_log', 'm_o_ccn_out')
disp('Ready')
```

APPENDIX C

CORTICO-HIPPOCAMPAL LESION MODEL CODE

```
% Cortico-Cerebellar Network (Lesion Model). Acquisition Case.
%
% This code simulates lesion model for acquisition case.
%
% After network initialization two phases exist in this code:
% 1) Adapt network to context
% 2) Adapt network for acquisition
%
% Algorithm
%
% - Initialize Network (# of layers, neurons, weights, biases)
%
% - Adapt Network
%
%       Minus Phase:
%       1) Clamp only external input to the input units
%       2) Settle the network with the settling routine
%
%       Plus Phase:
%       3) Clamp one training vector (external input and target) to the visible
%      units of the network
%       4) Settle the network with the settling routine
%          (only changes activations of units, not the weights), therefore get
%      the activations
%
%       Weight Change:
%       5) Calculate and apply appropriate weight changes
%          perform "Soft Weight Bounding"
%          calculate individual mean square errors
%       6) Repeat steps 1 through 5 for all training vectors (1 epoch)
%          calculate overall mean square errors for one epoch
%       7) Repeat steps 1 through 6 until the mean square error is below a
%      threshold
%
%   Settling Routine(synchronous updating):
%   1) Force the outputs of all visible units to the specified input vector
%   2) Assign "0" output value to all unknown units
%   3) For all units: compute raw_netin = sum sending_act * weight
%      (Add both top-down input dot products and bottom-up input dot products)
%   4) For all units compute:
%
%       netin = netin + dt * (raw_netin - netin)
%       act = sigmoid(netin)
%
```

```
%   5) Repeat steps 3 to 4 for several processing cycles, until activation
%      changes go below a threshold

% Ilim Cagiran, 2006
% Revision: 08_i

clear
tic

load trn_data02.mat
% CREATE NETWORK DATA
%-------------------------------------------------
Si= 18; % Size of Input vector
% S1_hpn= 10; % Size of Hidden Layer of HPN
% S2_hpn= 19; % Size of Output Layer of HPN
S1_ccn= 60; % Size of Hidden Layer of CCN
S2_ccn= 1; % Size of Output Layer of CCN

% ----------------------
% Due to "soft weight bounding" weight range is (0,1). Therefore weight
% values are centered on the middle value of 0.5 instead of 0.

% Intialize CCN
U_ccn_lo= 0.3; %U LOwer. Due to SWB, max value of U can only be 0.5
U_ccn_up= 0.1; %U UPper.
% WEIGHTS: The indices in "W_J_K" designates the weight matrix from the J'th
% layer to the K'th layer in this code.
% Input layer is the 1st layer. Hidden layer is the 2nd layer.
% Output layer is the 3rd layer.
% "W_ccn_2e_3i" designates the weight matrix from the 2nd Excitatory layer to
% the 3th Inhibitory layer in this code.
W_ccn_1e_2e= U_ccn_lo*rands(S1_ccn,Si)+0.5; % rands: returns an S-by-R weight
% matrix of random values between -1 and 1

W_ccn_2e_3e= U_ccn_up*rands(S2_ccn,S1_ccn)+0.5;
W_ccn_3e_2e= W_ccn_2e_3e';

% Biases
b_ccn_2= zeros(S1_ccn,1);
b_ccn_3= zeros(S2_ccn,1);

sc_e= 0.3; % dt. temporal integration constant for Excitatory neurons.
sc_i= 0.4;  % dt. temporal integration constant for Inhibitory neurons
scale_ff= 1 ; % overall weight scale for FeedForward inhibition. multiplied
% with weights for different layer level excitatory to inhibitory connections
scale_fb= 1   ; % overall weight scale for FeedBack inhibition. multiplied
% with weights for same layer level excitatory to inhibitory connections.
scale_ff_up= 1 ;
```

```
scale_fb_up= 1 ;
w_mult_2e_3e= 1 ;
w_mult_1e_2e= 1 ;

lr_mult_ccn= 1;
% ----------------------
scrsz = get(0,'ScreenSize');
figure('Position',[scrsz(3)/10 scrsz(4)/10 scrsz(3)/1.33 scrsz(4)/1.33])

% ADAPT NETWORK TO CONTEXT
%-----------------------------------------------
n_ep= 300;
mse_log=zeros(n_ep,5); % to improve program performance
disp('Please wait')
%
dwt_thresh_ccn= 1.e-3; % weight change threshold. default is 0.1
[r,c]=size(inpt_cntx);
ki2=50; % after how many epochs results will be showed
ki2=ceil(n_ep/(ki2*30))*ki2;
for ko2=1:n_ep
   mse_o_hpn = 0; mse_o_ccn = 0; mse_thr_o_hpn = 0; mse_thr_o_ccn = 0;
   m_o_aL3_ccn_m = 0;  m_o_aL2_ccn_m = 0;
         for cnt2=1:c
      aL1_hpn= inpt_cntx(:,cnt2); y_hpn= hpn_out_cntx(:,cnt2);
      aL1_ccn= inpt_cntx(:,cnt2); y_ccn= ccn_out_cntx(:,cnt2);
      if y_hpn(end)==1    % conditional response target is 1
         lr_hpn= 0.25; % Learning Rate hpn
         lr_ccn_lo= lr_mult_ccn*0.1; % Learning Rate LOwer layer
         lr_ccn_up= lr_mult_ccn*0.5; % Learning Rate UPper layer
      elseif y_hpn(end)==0
         lr_hpn= 0.025;
         lr_ccn_lo= lr_mult_ccn*0.01;
         lr_ccn_up= lr_mult_ccn*0.05;
      else
         error('Conditional Response Target should be either 0 or 1')
      end

      % MINUS PHASE
      %-----------------------

      % CCN OUTPUT CALCULATION / STARTS - MINUS PHASE
      %-----------------------------------------------
      % Assign "0" output value to all unknown units
      aL2_ccn= zeros(S1_ccn,1);
      aL3_ccn= zeros(S2_ccn,1);

      ninL2_ccn= zeros(S1_ccn,1); % NetIN, initial values
      ninL3_ccn= zeros(S2_ccn,1);
```

```
warning off MATLAB:divideByZero
delta_aL2_ccn=1; delta_aL3_ccn=1;
while (delta_aL2_ccn > dwt_thresh_ccn) | ...
      (delta_aL3_ccn > dwt_thresh_ccn)
   aL2_ccn_pr= aL2_ccn; aL3_ccn_pr= aL3_ccn; % _pr means Previous.
   % used to calculate change.

   % Add both bottom-up & top-down input dot products.
   % For inhibitory connections use "-".
   nL2_ccn= w_mult_1e_2e*(W_ccn_1e_2e*aL1_ccn)+ ...
      W_ccn_3e_2e*aL3_ccn + b_ccn_2; % Layer 2e
   nL3_ccn= w_mult_2e_3e*W_ccn_2e_3e*aL2_ccn + b_ccn_3; % Layer3e
   % output units are not self connected. No top-down input.

   % netin = netin + dt * (raw_netin - netin)
   ninL2_ccn= ninL2_ccn + sc_e*(nL2_ccn - ninL2_ccn);
   ninL3_ccn= ninL3_ccn + sc_e*(nL3_ccn - ninL3_ccn);

   aL2_ccn= 1 ./ (1 + exp(-ninL2_ccn)); % logsigmoid
   aL3_ccn= 1 ./ (1 + exp(-ninL3_ccn)); % logsigmoid

   % average of activations change ratio
   delta_aL2_ccn= mean(abs((aL2_ccn-aL2_ccn_pr)./aL2_ccn_pr));
   delta_aL3_ccn= mean(abs((aL3_ccn-aL3_ccn_pr)./aL3_ccn_pr));

end
aL2_ccn_m= aL2_ccn; aL3_ccn_m= aL3_ccn; % _m, Minus
if y_hpn(end)==1    % conditional response target is 1
   cr_ccn(ko2) = aL3_ccn_m; % Conditional Response ccn
end

% PLUS PHASE
%-----------------

% CCN OUTPUT CALCULATION / STARTS - PLUS PHASE
%----------------------------------------------
% Assign "0" output value to all unknown units
aL2_ccn= zeros(S1_ccn,1);
aL3_ccn= y_ccn;

ninL2_ccn= zeros(S1_ccn,1); % NetIN, initial values

warning off MATLAB:divideByZero
delta_aL2_ccn=1;
while delta_aL2_ccn > dwt_thresh_ccn
   % _pr means Previous. used to calculate change.
   aL2_ccn_pr= aL2_ccn;
```

```
    % Add both bottom-up & top-down input dot products.
    % For inhibitory connections use "-".
    nL2_ccn= w_mult_1e_2e*(W_ccn_1e_2e*aL1_ccn)+ ...
       W_ccn_3e_2e*aL3_ccn + b_ccn_2; % Layer 2e

    % netin = netin + dt * (raw_netin - netin)
    % act = sigmoid(netin)
    ninL2_ccn= ninL2_ccn + sc_e*(nL2_ccn - ninL2_ccn);

    aL2_ccn= 1 ./ (1 + exp(-ninL2_ccn)); % logsigmoid

    % average of activations change ratio
    delta_aL2_ccn= mean(abs((aL2_ccn-aL2_ccn_pr)./aL2_ccn_pr));

end
aL2_ccn_p= aL2_ccn; aL3_ccn_p= y_ccn; % _p, Plus


% CALCULATE WEIGHT CHANGES
%-------------------------
% delta_W_J_K= learning rate*(x_J_plus*y_K_plus - x_J_minus*y_K_minus)
% sending unit with activation x_J, receiving unit with activation y_K

% in "minus phase" aL1 is clamped, in "plus phase" aL1 and aL3 are
% both clamped

W_ch_ccn_1e_2e=zeros(S1_ccn,Si);
for cnt3=1:Si
    W_ch_ccn_1e_2e(:,cnt3)= lr_ccn_lo*( aL1_ccn(cnt3)*aL2_ccn_p ...
        - aL1_ccn(cnt3)*aL2_ccn_m );
end
W_ccn_1e_2e= W_ccn_1e_2e + W_ch_ccn_1e_2e; % weight update
% --------------
W_ch_ccn_2e_3e=zeros(S2_ccn,S1_ccn);
for cnt3=1:S1_ccn
    W_ch_ccn_2e_3e(:,cnt3)= lr_ccn_up*( aL2_ccn_p(cnt3)*aL3_ccn_p ...
        - aL2_ccn_m(cnt3)*aL3_ccn_m );
end
W_ccn_2e_3e= W_ccn_2e_3e + W_ch_ccn_2e_3e; % weight update
% --------------
W_ch_ccn_3e_2e=zeros(S1_ccn,S2_ccn);
for cnt3=1:S2_ccn
    W_ch_ccn_3e_2e(:,cnt3)= lr_ccn_up*( aL3_ccn_p(cnt3)*aL2_ccn_p ...
        - aL3_ccn_m(cnt3)*aL2_ccn_m );
end
W_ccn_3e_2e= W_ccn_3e_2e + W_ch_ccn_3e_2e; % weight update
% --------------
```

```
   % CCN BIAS UPDATE
   b_ccn_2= b_ccn_2 + lr_ccn_lo*(aL2_ccn_p-aL2_ccn_m);
   b_ccn_3= b_ccn_3 + lr_ccn_up*(aL3_ccn_p-aL3_ccn_m);

   % mean square error individual
   mse_i_ccn= sum( ( y_ccn - aL3_ccn_m ).^2 )/S2_ccn;
   % mean square error individual. thresholded, x=0 if x<.5 & x=1 if x>=.5
   mse_thr_i_ccn= sum( ( y_ccn - round(aL3_ccn_m) ).^2 )/S2_ccn;
   %-----------------------
   mse_o_ccn = mse_o_ccn + mse_i_ccn; % mse overall for one epoch
   % mse_thr overall for one epoch
   mse_thr_o_ccn = mse_thr_o_ccn + mse_thr_i_ccn;
   %-----------------------
   % mean values are required for bar graphs
   m_o_aL3_ccn_m = m_o_aL3_ccn_m + aL3_ccn_m;

   m_o_aL2_ccn_m = m_o_aL2_ccn_m + aL2_ccn_m;

end

% mse log data: 1st column is # of epoch, 2nd is mse_o_hpn,
% 3rd is mse_o_ccn, 4th is mse_thr_o_hpn, 5th is mse_thr_o_ccn
mse_o_hpn = mse_o_hpn/c; mse_o_ccn = mse_o_ccn/c;
mse_thr_o_hpn= mse_thr_o_hpn/c; mse_thr_o_ccn= mse_thr_o_ccn/c;
mse_log(ko2,1)=ko2; mse_log(ko2,2)=mse_o_hpn; mse_log(ko2,3)=mse_o_ccn;
mse_log(ko2,4)=mse_thr_o_hpn; mse_log(ko2,5)=mse_thr_o_ccn;
% mean values are required for bar graphs
m_o_aL3_ccn_m = m_o_aL3_ccn_m/c;  m_o_aL2_ccn_m = m_o_aL2_ccn_m/c;

if rem(ko2,ki2)==0 | ko2==1 | ko2==n_ep

   dtxt3=num2str(mse_o_hpn);
   dtxt3b=num2str(mse_thr_o_hpn);
   dtxt4=num2str(ko2);
   %---------------
   dtxt5=num2str(mse_o_ccn);
   dtxt5b=num2str(mse_thr_o_ccn);

   dtxtg7=['mse_ccn=', dtxt5,'  &  mse_thr_ccn=', dtxt5b,...
         '  at epoch ', dtxt4];

   disp(dtxtg7)
end
subplot(2,3,2), bar(m_o_aL3_ccn_m), axis([0,2,0,1]), ...
   TITLE('Output Layer Activity ')
subplot(2,3,5), bar(m_o_aL2_ccn_m), axis([0,61,0,1]), ...
   TITLE('Hidden Layer Activity ')
```

```
   M(ko2) = getframe(gcf);
end
toc

% figure, movie(M)

% Plot Performance of Network
figure
semilogy(mse_log(:,1), mse_log(:,2), 'b--',mse_log(:,1),mse_log(:,3),'g--',...
        mse_log(:,1), mse_log(:,4), 'b-', mse_log(:,1),mse_log(:,5),'g-')
TITLE('PERFORMANCE PLOT, GENEREC, SWB')
XLABEL('EPOCHS')
YLABEL('MSE, blue:hpn, green:ccn, solid:thresholded, dashed:raw')

figure('Position',[scrsz(3)/10 scrsz(4)/10 scrsz(3)/1.33 scrsz(4)/1.33])

% ADAPT NETWORK
%-----------------------------------------------
n_ep= 300;
mse_log=zeros(n_ep,5); % to improve program performance
disp('Please wait')
%
dwt_thresh_ccn= 1.e-3; % weight change threshold. default is 0.1
[r,c]=size(inpt);
ki2=50; % after how many epochs results will be showed
ki2=ceil(n_ep/(ki2*30))*ki2;
for ko2=1:n_ep
   mse_o_hpn = 0; mse_o_ccn = 0; mse_thr_o_hpn = 0; mse_thr_o_ccn = 0;
   m_o_aL3_ccn_m = 0;  m_o_aL2_ccn_m = 0;
         for cnt2=1:c
      aL1_hpn= inpt(:,cnt2); y_hpn= hpn_out(:,cnt2);
      aL1_ccn= inpt(:,cnt2); y_ccn= ccn_out(:,cnt2);
      if y_hpn(end)==1    % conditional response target is 1
         lr_hpn= 0.25;   % Learning Rate hpn
         lr_ccn_lo= lr_mult_ccn*0.1; % Learning Rate LOwer layer
         lr_ccn_up= lr_mult_ccn*0.5; % Learning Rate UPper layer
      elseif y_hpn(end)==0
         lr_hpn= 0.025;
         lr_ccn_lo= lr_mult_ccn*0.01;
         lr_ccn_up= lr_mult_ccn*0.05;
      else
         error('Conditional Response Target should be either 0 or 1')
      end

      % MINUS PHASE
      %-----------------------

      % CCN OUTPUT CALCULATION / STARTS - MINUS PHASE
```

```
%-----------------------------------------------
% Assign "0" output value to all unknown units
aL2_ccn= zeros(S1_ccn,1);
aL3_ccn= zeros(S2_ccn,1);

ninL2_ccn= zeros(S1_ccn,1); % NetIN, initial values
ninL3_ccn= zeros(S2_ccn,1);

warning off MATLAB:divideByZero
delta_aL2_ccn=1; delta_aL3_ccn=1;
while (delta_aL2_ccn > dwt_thresh_ccn) | ...
      (delta_aL3_ccn > dwt_thresh_ccn)
   aL2_ccn_pr= aL2_ccn; aL3_ccn_pr= aL3_ccn; % _pr means Previous.
   % used to calculate change.

   % Add both bottom-up & top-down input dot products.
   % For inhibitory connections use "-".
   nL2_ccn= w_mult_1e_2e*(W_ccn_1e_2e*aL1_ccn)+ ...
      W_ccn_3e_2e*aL3_ccn + b_ccn_2; % Layer 2e
   nL3_ccn= w_mult_2e_3e*W_ccn_2e_3e*aL2_ccn + b_ccn_3; % Layer3e
   % output units are not self connected. No top-down input.

   % netin = netin + dt * (raw_netin - netin)
   ninL2_ccn= ninL2_ccn + sc_e*(nL2_ccn - ninL2_ccn);
   ninL3_ccn= ninL3_ccn + sc_e*(nL3_ccn - ninL3_ccn);

   aL2_ccn= 1 ./ (1 + exp(-ninL2_ccn)); % logsigmoid
   aL3_ccn= 1 ./ (1 + exp(-ninL3_ccn)); % logsigmoid

   % average of activations change ratio
   delta_aL2_ccn= mean(abs((aL2_ccn-aL2_ccn_pr)./aL2_ccn_pr));
   delta_aL3_ccn= mean(abs((aL3_ccn-aL3_ccn_pr)./aL3_ccn_pr));

end
aL2_ccn_m= aL2_ccn; aL3_ccn_m= aL3_ccn; % _m, Minus
if y_hpn(end)==1    % conditional response target is 1
   cr_ccn(ko2) = aL3_ccn_m; % Conditional Response ccn
end

% PLUS PHASE
%-----------------

% CCN OUTPUT CALCULATION / STARTS - PLUS PHASE
%-----------------------------------------------
% Assign "0" output value to all unknown units
aL2_ccn= zeros(S1_ccn,1);
aL3_ccn= y_ccn;
```

```
ninL2_ccn= zeros(S1_ccn,1); % NetIN, initial values

warning off MATLAB:divideByZero
delta_aL2_ccn=1;
while delta_aL2_ccn > dwt_thresh_ccn
   % _pr means Previous. used to calculate change.
   aL2_ccn_pr= aL2_ccn;

   % Add both bottom-up & top-down input dot products.
   % For inhibitory connections use "-".
   nL2_ccn= w_mult_1e_2e*(W_ccn_1e_2e*aL1_ccn)+ ...
      W_ccn_3e_2e*aL3_ccn + b_ccn_2; % Layer 2e

   % netin = netin + dt * (raw_netin - netin)
   % act = sigmoid(netin)
   ninL2_ccn= ninL2_ccn + sc_e*(nL2_ccn - ninL2_ccn);

   aL2_ccn= 1 ./ (1 + exp(-ninL2_ccn)); % logsigmoid

   % average of activations change ratio
   delta_aL2_ccn= mean(abs((aL2_ccn-aL2_ccn_pr)./aL2_ccn_pr));

end
aL2_ccn_p= aL2_ccn; aL3_ccn_p= y_ccn; % _p, Plus


% CALCULATE WEIGHT CHANGES
%-------------------------
% delta_W_J_K= learning rate*(x_J_plus*y_K_plus - x_J_minus*y_K_minus)
% sending unit with activation x_J, receiving unit with activation y_K

% in "minus phase" aL1 is clamped, in "plus phase" aL1 and aL3
% are both clamped

W_ch_ccn_1e_2e=zeros(S1_ccn,Si);
for cnt3=1:Si
   W_ch_ccn_1e_2e(:,cnt3)= lr_ccn_lo*( aL1_ccn(cnt3)*aL2_ccn_p ...
      - aL1_ccn(cnt3)*aL2_ccn_m );
end
W_ccn_1e_2e= W_ccn_1e_2e + W_ch_ccn_1e_2e; % weight update
% --------------

W_ch_ccn_2e_3e=zeros(S2_ccn,S1_ccn);
for cnt3=1:S1_ccn
   W_ch_ccn_2e_3e(:,cnt3)= lr_ccn_up*( aL2_ccn_p(cnt3)*aL3_ccn_p ...
      - aL2_ccn_m(cnt3)*aL3_ccn_m );
end
W_ccn_2e_3e= W_ccn_2e_3e + W_ch_ccn_2e_3e; % weight update
```

```
    % --------------

    W_ch_ccn_3e_2e=zeros(S1_ccn,S2_ccn);
    for cnt3=1:S2_ccn
       W_ch_ccn_3e_2e(:,cnt3)= lr_ccn_up*( aL3_ccn_p(cnt3)*aL2_ccn_p ...
          - aL3_ccn_m(cnt3)*aL2_ccn_m );
    end
    W_ccn_3e_2e= W_ccn_3e_2e + W_ch_ccn_3e_2e; % weight update
    % --------------
    b_ccn_2= b_ccn_2 + lr_ccn_lo*(aL2_ccn_p-aL2_ccn_m);
    b_ccn_3= b_ccn_3 + lr_ccn_up*(aL3_ccn_p-aL3_ccn_m);

    % mean square error individual
    mse_i_ccn= sum( ( y_ccn - aL3_ccn_m ).^2 )/S2_ccn;
    % mean square error individual. thresholded, x=0 if x<.5 & x=1 if x>=.5
    mse_thr_i_ccn= sum( ( y_ccn - round(aL3_ccn_m) ).^2 )/S2_ccn;
    %-----------------------
    % mse overall for one epoch
    mse_o_ccn = mse_o_ccn + mse_i_ccn;
    % mse_thr overall for one epoch
    mse_thr_o_ccn = mse_thr_o_ccn + mse_thr_i_ccn;
    %-----------------------
    % mean values are required for bar graphs
    m_o_aL3_ccn_m = m_o_aL3_ccn_m + aL3_ccn_m;
    m_o_aL2_ccn_m = m_o_aL2_ccn_m + aL2_ccn_m;

end

% mse log data: 1st column is # of epoch, 2nd is mse_o_hpn,
% 3rd is mse_o_ccn, 4th is mse_thr_o_hpn, 5th is mse_thr_o_ccn
mse_o_hpn = mse_o_hpn/c; mse_o_ccn = mse_o_ccn/c;
mse_thr_o_hpn= mse_thr_o_hpn/c; mse_thr_o_ccn= mse_thr_o_ccn/c;
mse_log(ko2,1)=ko2; mse_log(ko2,2)=mse_o_hpn; mse_log(ko2,3)=mse_o_ccn;
mse_log(ko2,4)=mse_thr_o_hpn; mse_log(ko2,5)=mse_thr_o_ccn;
% mean values are required for bar graphs
m_o_aL3_ccn_m = m_o_aL3_ccn_m/c;  m_o_aL2_ccn_m = m_o_aL2_ccn_m/c;

if rem(ko2,ki2)==0 | ko2==1 | ko2==n_ep

    dtxt3=num2str(mse_o_hpn);
    dtxt3b=num2str(mse_thr_o_hpn);
    dtxt4=num2str(ko2);
    %---------------
    dtxt5=num2str(mse_o_ccn);
    dtxt5b=num2str(mse_thr_o_ccn);
    dtxtg7=['mse_ccn=', dtxt5,'  &  mse_thr_ccn=', dtxt5b, ...
          '  at epoch ', dtxt4];
    disp(dtxtg7)
```

```
   end
   subplot(2,3,1), bar(cr_ccn(ko2)), axis([0,2,0,1]), ...
      TITLE('CR CCN Activity ')
   subplot(2,3,2), bar(m_o_aL3_ccn_m), axis([0,2,0,1]), ...
      TITLE('Output Layer Activity ')
   subplot(2,3,5), bar(m_o_aL2_ccn_m), axis([0,61,0,1]), ...
      TITLE('Hidden Layer Activity ')

   M(ko2) = getframe(gcf);
end
toc

% Plot Performance of Network
figure
semilogy(mse_log(:,1), mse_log(:,2), 'b--',mse_log(:,1),mse_log(:,3),'g--',...
       mse_log(:,1), mse_log(:,4), 'b-', mse_log(:,1),mse_log(:,5),'g-')
TITLE('PERFORMANCE PLOT, GENEREC, SWB')
XLABEL('EPOCHS')
YLABEL('MSE, blue:hpn, green:ccn, solid:thresholded, dashed:raw')

disp('mse_log will be saved as gr_ccn_????.mat ')
reply = input('Enter the last part of the file name (eg. trial2):  ','s');
fln=['gr_ccn_', reply];
save(fln, 'mse_log')
disp('Ready')
```

REFERENCES

1. Gluck, M.A. and Myers, C.E., *Gateway to Memory: An Introduction To Neural Network Modeling of The Hippocampus And Learning,* MIT Press, Massachusetts, 2001.

2. Dayan, P. and Abbot, L.F., *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, MIT Press, Massachusetts, 2001.

3. Hinton, Ge., and Sejnowski, TJ., "Learning and relearning in Boltzmann machines". In D.E. Rumelhart, and J.L. McClelland, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1, Foundations. Cambridge, MA, *MIT Press*,282-317, 1986.

4. O'Reilly, R.C. and Munakata, Y., *Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain*, Cambridge, MA, MIT Press, 2000.

5. Gluck, M.A., Meeter, M., and Myers, C.E., "Computational models of the hippocampal region: linking incremental learning and episodic memory", *Trends in Cognitive Sciences*, Vol.7, No.6, June 2003.

6. Gabriel, M., and Moore, J.W., *Learning and Computational Neuroscience*, MIT Press, Massachusetts, 1990.

7. Johnston, D., Wu, S.M., *Foundations of Cellular Neurophysiology*, MIT Press, Massachusetts, 1995.

8. Hertz, J., Krogh, A., and Palmer, R.G., *Introduction to the Theory of Neural Computation*, Addison-Wesley, CA, 1991.

9. Hebb, D.O., *The Organization of Behavior: A Neuropsychological Theory*, Wiley, New York, 1949.

10. Hinton, G.E., "Connectionist Learning Procedures," *Artificial Intelligence*, 40:185-234.

11. Goodall, M.C., “Performance of a Stochastic Net”, *Nature*, 185:557-558, 1960.

12. Sejnowski, T.J., “Storing covariance with nonlinearly interacting neurons”, *Journal of Mathematical Biology*, 4:303-321, 1977.

13. Oja, E., “A Simplified Neuron Model as a Principal Component Analyzer”, *Journal of Mathematical Biology*, 16:267-273, 1982.