Asymmetric Weight Matrices as a Method to Resolve Binocular Neuronal Activity in a Reward Time Learning Network

Victor Nguyen[1]
under the supervision of Harel Shouval[2]

[1]University of Washington - Seattle, WA
[2]University of Texas Medical School - Houston, TX

July 2009

Abstract

The exact mechanism of where and how our neurons come to represent our conscious perception of time (of the seconds scale) is unknown. Results by Shuler and Bear show that such a process might be possible within the primary visual cortex through operant conditioning. Gavornik *et al.* have shown that it is theoretically possible for such a network to arise through changes in only synaptic weights. Their model has been shown to work in both cases of purely monocular neurons and with overlapping binocular neurons. However, with binocular neurons, a modified Hebbian function is required to prevent cross contamination of non-stimulated neurons from being excited. This paper explores how the Hebbian function affects the weight matrix and its subsequent role on the activity of a binocular network within the Gavornik model. We heuristically find several Hebbian functions that correct for the original contamination problem and that negative cross subset weights are required to prevent non-stimulated neurons from activating.

Contents

List of Figures

Chapter 1

Introduction

1.1 General introduction[3]

Historically, the encoding of time has been believed to be operated by a single mechanism overseeing all time scales. In recent years, however, there has been some research and theories that support several different scales, with each scale being controlled by separate, but possibly overlapping, mechanisms. Currently, the mechanisms controlling milliseconds and seconds are not understood well. Encoding of seconds is of particular interest because it forms the basis of our conscious perception of time.

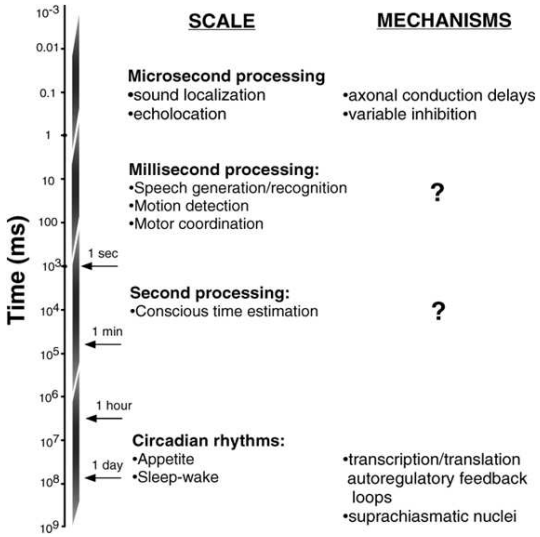


Figure 1.1: Time scale

Several theories favor decentralized state-dependent networks as the mechanism governing such processes (among these are Buonomano & Merzenich 1995[4], Deisz & Prince 1989[5], Newberry & Nicoll 1984[6]). These bottom-up theories rely on the (synaptic) plasticity of the network. When a stimulus arrives in a network, it activates a subset of neurons and inhibits others. If a second stimulus arrives before all activity has returned to resting states, the stimulus encounters a network in a differing state. This may continue indefinitely. Thus, each network state may come to signify a different portion of the stimulus. Of the properties represented, time may be one of them. These different states altered by stimulation may be modified and refined through learning and reinforcement causing changes in the synaptic weights (the exact mechanism of change, whether it be through concentration of neurotransmitters, number of receptors, etc, varies by theory). Thus, a specific neuronal state can encode for a certain, complex temporal pattern.

1.2 Learning of reward signals in rat primary visual cortex as found by Shuler and Bear[7]

While evidence of disperse locations of time encoding are relatively rare, Shuler and Bear have generated evidence that temporal processing might occur in a primary sensory cortex rather than requiring a higher level brain area like the cerebral cortex. They found that neurons in the rat primary visual cortex are able to undergo a change to encode the timing of a reward signal. In naïve networks, activity persists during stimulation and decays quickly after input is removed. In trained networks, excitatory and inhibitory neuronal activity persisted until the expected time of reward depending on the specific stimulus. These results point towards a possible mechanism of the conscious perception of time as prediction requires the correct tracking of time progression. While other experimental results are sparse, there is some evidence indicating that this sort of persistent activity allows the maintenance of a stimulus in working memory (with the possibility of timing being a property of the stimulus[8,9]

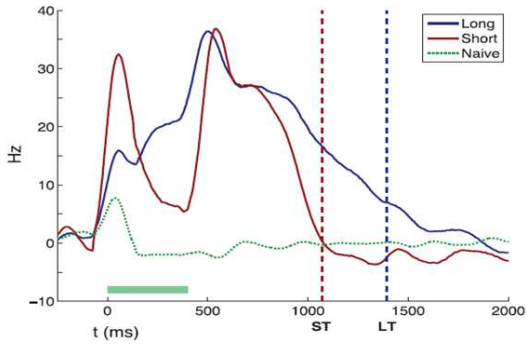


Figure 1.2: In the naïve network, activity persists only during stimulation. In trained networks, activity can persist until time of reward.

Rats were fitted with goggles that provided full field illumination in each eye separately. Left eye stimulation meant the rat needed to lick a water tube for water "x" number of times (x usually equaled 6) for water, whereas right eye stimulation meant the rat needed "2x" licks (the number of licks were mapped on to a specific time interval in the experiment). Only half the trials were rewarded to determine if the learning occurred as a direct result of the reward or the reward prediction/expectancy. Electrodes were chronically implanted in area V1 to measure neuronal activity before, during, and after stimulation.

1.3 Modeling Shuler and Bear's results theoretically[10]

Isolated neurons have a natural time constant with which they decay accordingly after stimulation. This time constant is relatively short so that neuronal activity does not last long after stimulation. If, however, a neuron is connected to others within a network capable of feedback, the decay can be altered through reverberatory propagation. For instance, if we stimulate two neurons, they stimulate each other in turn. Thus, activity persists for a short duration longer after receiving mutual stimulation from the other neuron.

Through a recurrent cortical network that undergoes synaptic plasticity according to a Hebbian process (i.e. coactive neurons become more strongly connected while non-coactive neuron connections become weaker), Gavornik et al. show it is possible for a network to learn specific time intervals from specific stimulus input patterns. Depending on which neurons (each corresponding to an eye) are stimulated, changes in the strength of synaptic connections, called weights, allow for neuronal activity to persist until the time of reward.

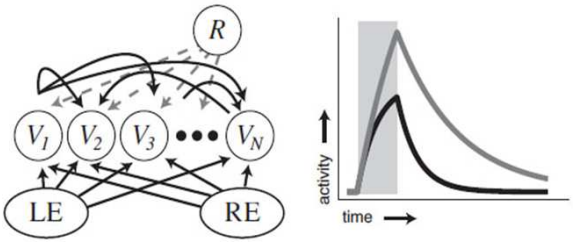


Figure 1.3: Left figure: Each eye projects to neurons in a recurrent layer. Reward signal (R) projects to all. Right figure: Example of single neuron activity during stimulation (grey box). Untrained neuron, in black, decays quickly while a neuron within a network with small lateral synaptic weights, in grey, is active longer.

According to this model, synaptic strengths are only allowed to change at the time of reward presentation. But the reward is presented seconds after brief stimulation activates the neurons for only milliseconds. How does the network "remember" if neurons were activated during stimulation then? Information regarding coincident activation of presynaptic and postsynaptic neurons is maintained in a temporary protoweight matrix. These entries, which individually represent the synaptic strength between exactly two neurons, grow according to a Hebbian function $V_{presynaptic} * V_{postsynaptic}$ (if the two neurons are coactive, their product is positive and the synaptic strength increases) and decay according to a prototime constant. At the time of reward, the weight matrix changes proportionally to the protoweight matrix.

(See section 2, Methods, for more details about the mathematical implementation of this model.) The model assumes that ongoing cortical activity is a natural inhibitor to potentiation during the time protoweights are converted to permanent weights. If a presynaptic neuron and postsynaptic neuron are active within the same time window (and in this case, it would be to the same stimulus), the Hebbian function is positive, as are the protoweights. Assuming cortical activity is not high enough to inhibit potentiation, the synaptic weights between those neurons increases. In the next epoch, the effective decay constant of the postsynaptic neuron is longer. Thus, the weight matrix is continually modified to produce a decay of a certain duration. Inhibitory cortical activity ensures that learning does not increase beyond that time as it decreases the synaptic weights if activity is too high at the time of reward (i.e. activity is persisting beyond reward time).

Under monocular stimulation patterns, this model responds and tracks correct growth robustly. There is a clear distinction between the neurons of each eye they effectively act as two completely different networks.

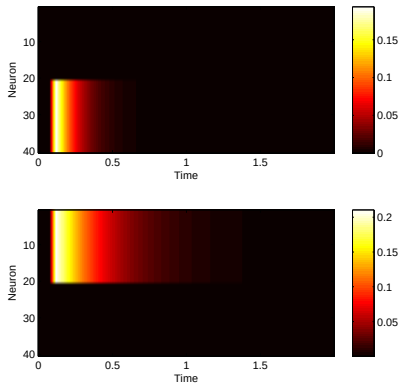


Figure 1.4: Monocular stimulation produces clear separation and correct timing

But what happens, then, when some of these neurons become binocular i.e. respond to stimulation of either eye? If these binocular neurons are stimulated strongly enough, they will begin to activate both sets of neurons despite one eye possibly not being activated initially. Thus, there will be an overlap of firing between the monocular networks. For example:

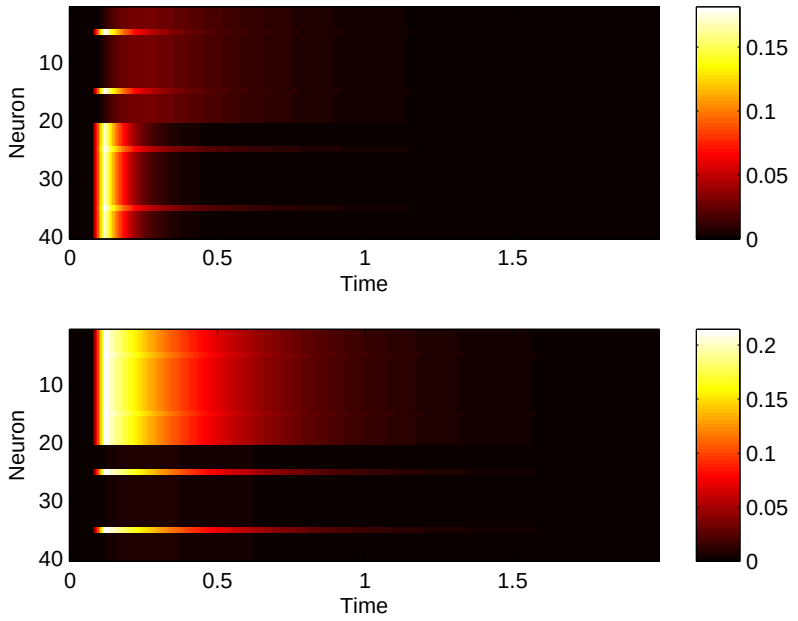


Figure 1.5: Binocular stimulation produces overlap and incorrect timing if the Hebbian function is not corrected

To separate the activity of the two populations, a change to the basic Hebbian growth function, $V_i(t) \cdot V_j(t)$ is required. Gavornik et al. were able to find a general form, $(V_i - \theta_1) * (V_j - \theta_1) - \theta_2$, that corrected this error. θ is a user defined constant. Heuristically, θ_1 was found to be 2.5 and θ_2 was 0.

The purpose of this project is to use the model developed by Gavornik *et al.* (2008) and explore different forms of the Hebbian plasticity function that allow for correct separation of the neuronal subsets even with binocularly active neurons as well as correct convergence to the reward signal. We found that such corrected forms of the Hebbian function resulted in asymmetric weight matrices.

Chapter 2

Methods

Using the information provided in Gavornik *et al.*s supporting information[10], the following three functions were implemented in Matlab 7.3.0.267 (R2006b):

Activity of neuron i over one epoch: **(1)** $\tau_m \frac{dV_i}{dt} = -V_i + I + \sum_N L_{ij} V_j$

where V_i represents the activity of neuron i on an arbitrary scale,
I is the input pattern,
L_{ij} is the weight matrix for presynaptic cell j and postsynaptic cell i,
τ_m is the intrinsic time constant,

Neuronal activity rises exponentially according to τ_m during stimulation and will decay according the values of L_{ij}.

Potentiation of protoweights over one epoch: **(2)** $\tau_p \frac{dL^p_{ij}}{dt} = -L^p_{ij} + H(V_i, V_j)$

where L^p_{ij} is the proto-weight matrix,
τ_p is the proto-weight time constant,
$H(V_i, V_j)$ is the Hebbian function,

To bridge the time gap between stimulation and reward, coactivation of neurons is recorded in a protoweight matric that also decays exponentially.

Potentiation of weights at time of reward: **(3)** $\frac{dL_{ij}}{dt} = \eta L^p_{ij}(t) \cdot (r_0 - \beta \frac{1}{N} \sum_{i=1}^{N} V_i(t)) \delta(t - T^\mu)$

where η is the learning rate,
β is a scaling factor that controls reward inhibition,
$\delta(x)$ is the Dirac delta function,
t is the time,
T^μ is the time of reward (μ representing of a certain pattern),
r_0 is the reward magnitude.

At the time of reward, protoweights are expressed in to the actual weight matrix and are regulated by constants η, β, and r_0.

Neuronal activity is calculated by a first order ordinary differential equation. This equation uses linear passive integrator neurons. Equations are solved in a forward Euler method.

Networks consisted of 40 neurons trained over 100 epochs on a 2s period. Neurons 1-20, 25, and 35 (simulating one eye, hereafter referred to as subset 1) were rewarded at time 1 s and neurons 5, 15, 21-40 (simulating the other eye, hereafter referred to as subset 2) learned reward time .5s. Rewards were given every trial. Each eye was stimulated every other epoch (50 epochs for each eye). Both L_{ij} and L^p_{ij} are initially set to 0. τ_m is set to .06 and τ_p is set to 5. Input of magnitude .3 is delivered from .075s to .125s.

Working parameters and functions were found heuristically. Results are presented visually using the Matlab functions imagesc and mesh. The actual Matlab code used is given in the Appendix.

Chapter 3

Results

Five forms of the Hebbian function were found to separate the two sets of neurons. However, not all forms successfully trained neurons to the correct time. As per the original model, we did not attempt an exact mathematical definition of reward time learning. Thus, we accept activity persisting until approximately the reward times as a sufficient model for the ability of the network to learn temporal representations.

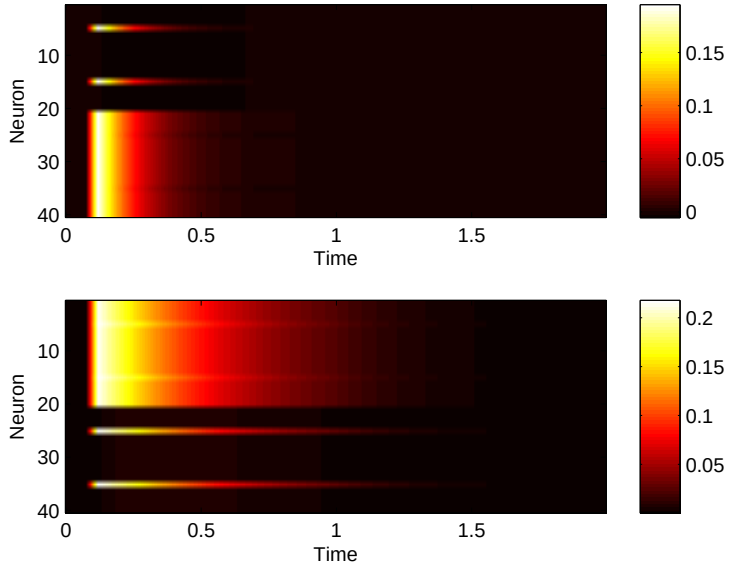
Activity and weight matrices at the end of training are presented in section **3.1**.
General trends found regarding the weight matrices are presented in section **3.2**.
Tests to determine the weight matrix' effect on neuronal activity are presented in section **3.3**.
Sample population activities for a purely monocular network and another with binocular neurons are presented in section **3.4**.

In sections **3.1** and **3.3**, the manipulated equations are listed at the beginning of each subsection. V_i represents the postsynaptic neuron and V_j represents the presynaptic neuron. $*$ denotes matrix multiplication. $\cdot$ denotes an array multiplication (element by element).

3.1 Neuronal activity and weight matrices at the end of training

3.1.1 $H(V_i, V_j) = V_i * (V_j - \theta_1) - \theta_2$
$\eta = 2.2, r_0 = .3, \beta = 30, \theta_1 = .013, \theta_2 = .005$

Figure 3.1: Neural activity for network 1



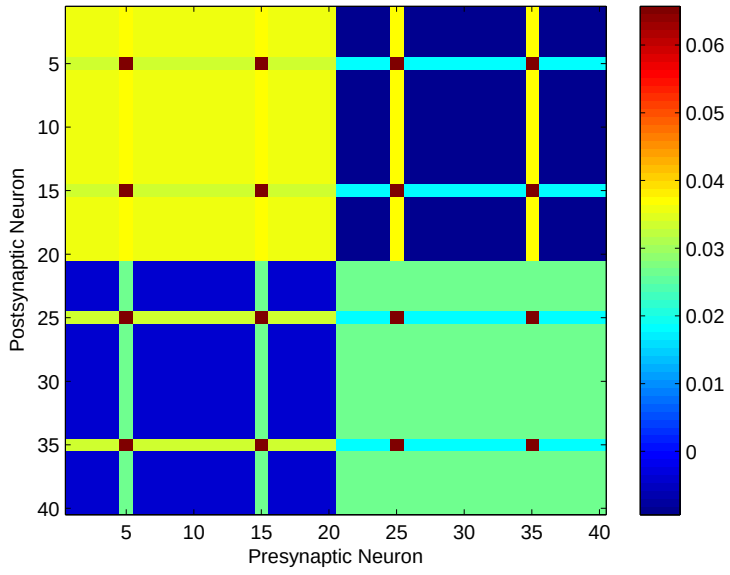


Figure 3.2: Weight matrix for network 1

Separation of the two eyes is nearly complete. There is a slight trace of subset 2 activity during subset 1 stimulation. Both subsets are trained to the correct times.

3.1.2 $H(V_i, V_j) = (V_i - \theta_1) \cdot ((V_i - \theta_1) > 0)) * (V_j - \theta_1) - \theta_2$
$\eta = 2.2, r_0 = .3, \beta = 30, \theta_1 = 0, \theta_2 = .04$

Figure 3.3: Neural activity for network 2

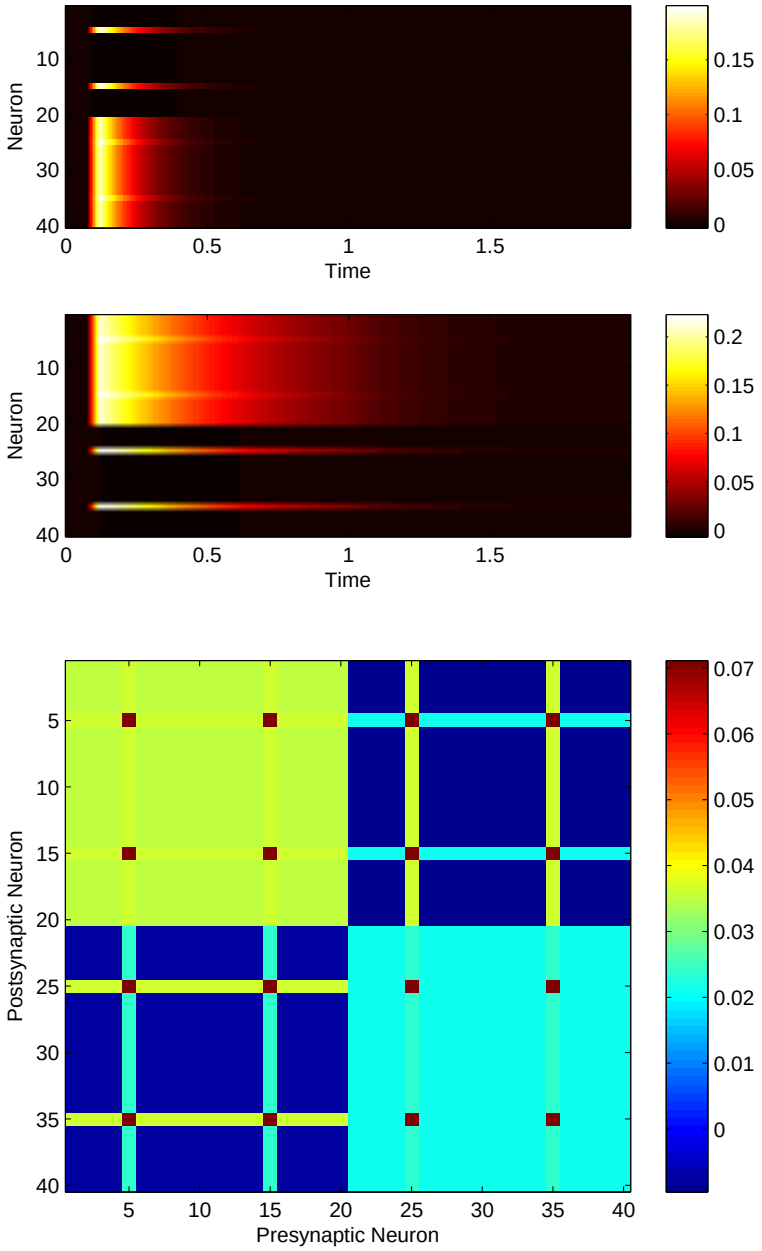
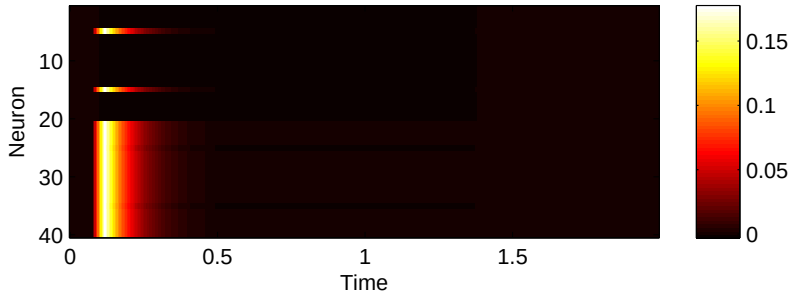


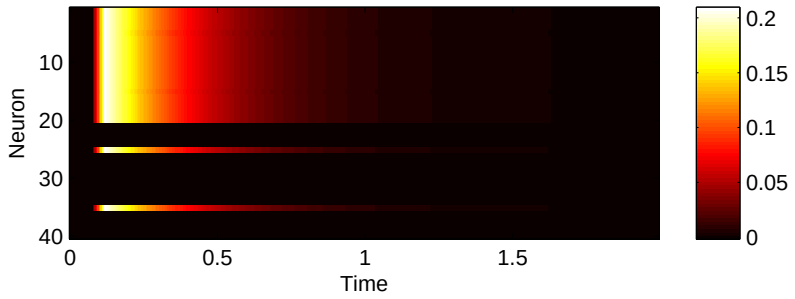
Figure 3.4: Weight matrix for network 2

Separation of the two eyes is complete. Only subset 2 is trained to the correct time (.5s). Subset 1 overshoots its reward timing by nearly 50%. It seems as if at about .5s, subset 1 becomes restimulated which could be a reason for the overshooting.

3.1.3 $H(V_i, V_j) = (V_i - \theta_1) \cdot ((V_i - \theta_1) > 0)) * (V_j - \theta_1) - \theta_2$
$\eta = 1, r_0 = .34, \beta = 1, \theta_1 = .023, \theta_2 = 0$

Figure 3.5: Neural activity for network 3





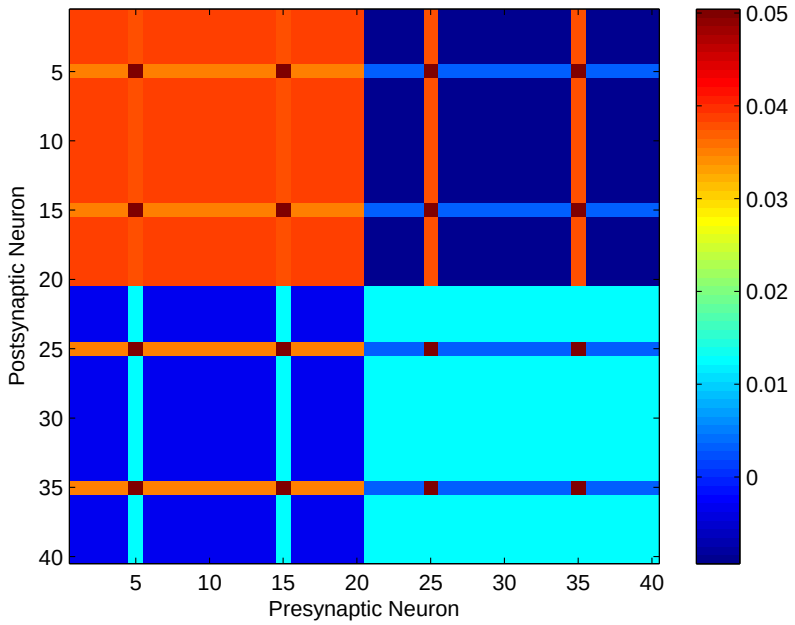
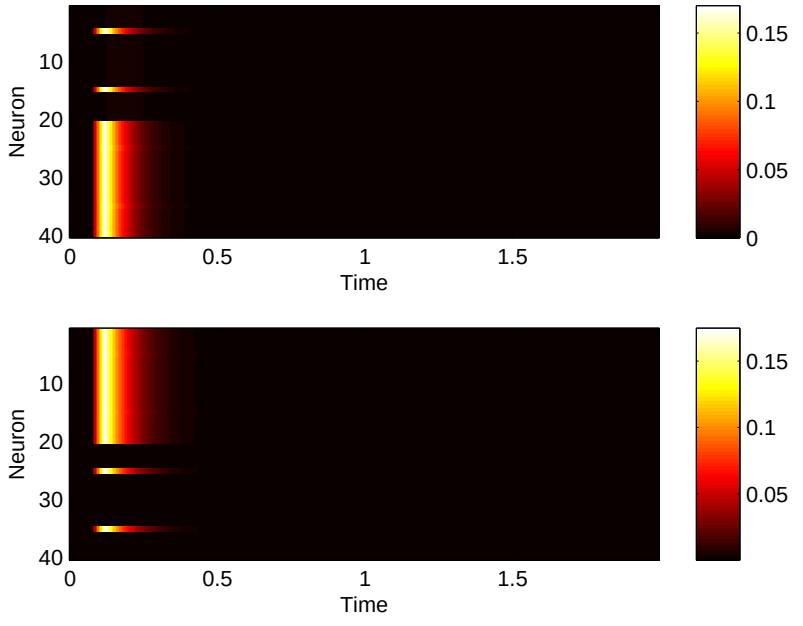


Figure 3.6: Weight matrix for network 3

Separation of the two eyes is complete. Both eyes are also trained to the correct times.

3.1.4 $H(V_i, V_j) = (V_i - \theta_1) \cdot ((V_i - \theta_1) > 0)) * ((V_j - \theta_1) \cdot ((V_j - \theta_1) > 0))\theta_2$
$\eta = 2.2, r_0 = .3, \beta = 30, \theta_1 = .08, \theta_2 = 0$

Figure 3.7: Neural activity for network 4



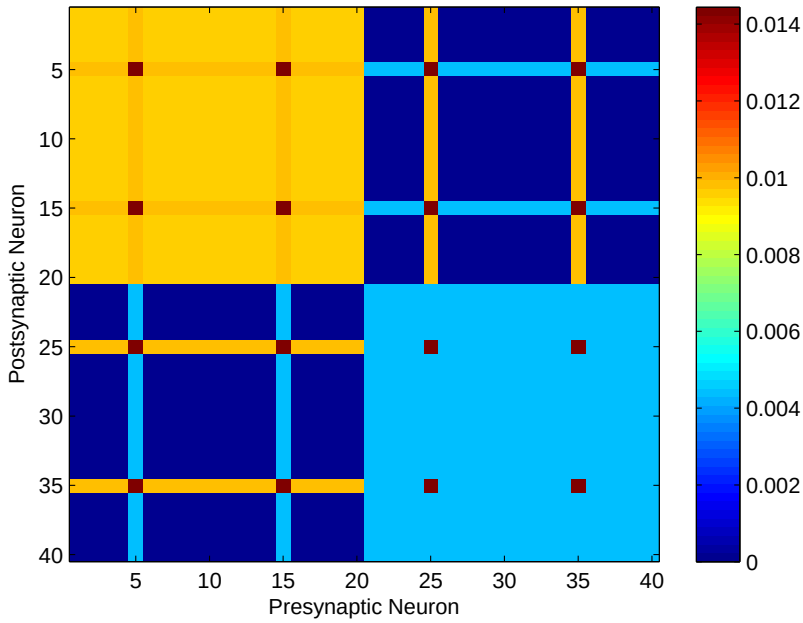
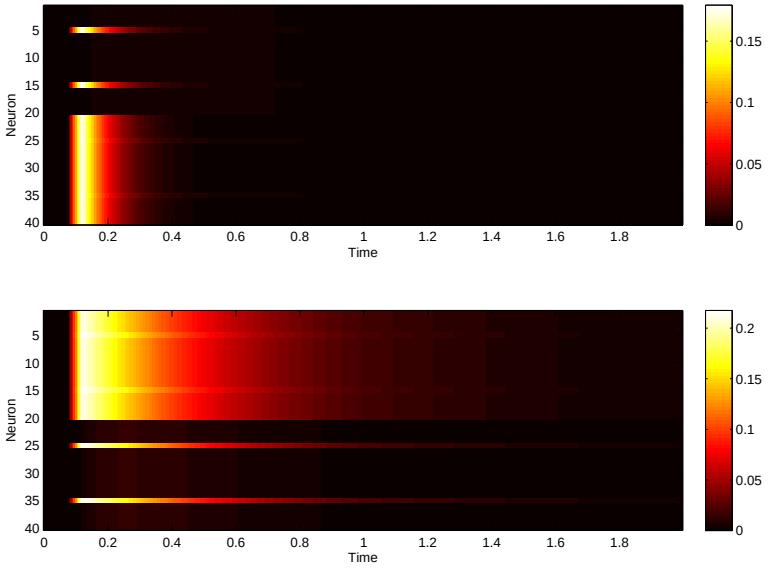


Figure 3.8: Weight matrix for network 4

Separation of the two eyes is complete. The two subsets have not been trained to the correct times. Both subsets undershoot the reward times decay to about the same time.

3.1.5 $H(V_i, V_j) = V_i * ((V_j - \theta_1) \cdot (V_j > 0)) - \theta_2$
$\eta = .73, r_0 = .37, \beta = 1, \theta_1 = .02, \theta_2 = 0$

Figure 3.9: Neural activity for network 5



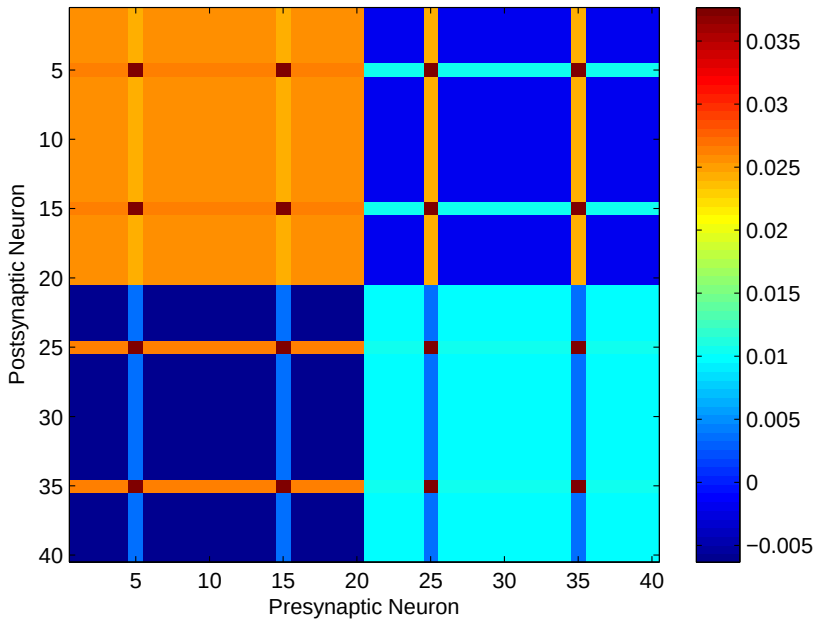


Figure 3.10: Weight matrix for network 5

Separation of the two eyes is nearly complete. Slight trace of subset 1 activity during subset 2 stimulation. Neither eye is trained to the correct times. Subset 1 overshoots its timing while subset 2 undershoots.

3.2 General trends found about the final weight matrices after training

If the weight matrices are divided into the four commonly recognized quadrants (one is in the top right corner, moving counter-clockwise), we see that:

- Weights are strongest in quadrant 2, followed by 4, 3, and finally 1
- Weights from a binocular neuron to itself are greater than any other weight
- Weights from a binocular neuron are always positive
- The more positive the weights in quadrants 2 and 4, the longer activity persists for those neurons
- Weights from binocular neurons to the other cells are stronger than vice versa except in quadrant 3
- In separated networks, cross subset monocular neuron weights are negative but neuronal activity is not
- In separated networks, cross subset monocular neuron weights are much more negative than weights to binocular neurons.
- In non-separated networks, binocular neuron weights are about 4 times that of monocular weights

3.3 Manually manipulating weight matrix entries and neuronal activity

While we have found some Hebbian functions that correct for binocular stimulation, we have not found *how* or *why* they work beyond an elementary understanding that they change the weight matrix. How does the weight matrix directly affect the neuronal activity? We explore this question in this section.

neuronal activity at the end of training is presented after manually manipulating the weight or protoweight matrix each epoch. The Hebbian function is kept at $V_i * V_j$. Under the title of each subsection, the Matlab code implemented is shown.

3.3.1 Preventing growth of cross subset weights

What happens if we prevent the growth of cross subset weights? How are they affecting the separation?

```
L_p(1:20,21:40) = 0;
L_p(21:40,1:20) = 0
```

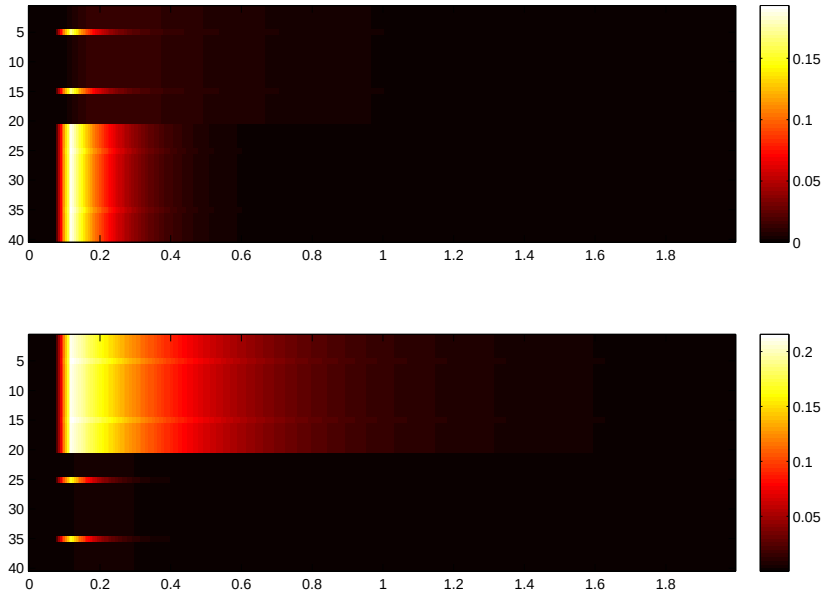


Figure 3.11: Neural activity after setting cross subset weights to 0

These parameters provide some separation - more than the original no-corrections scenario. However, the main problem is that the binocular neurons are no longer trained to the correct times. There is slight overlap in activity in the top panel. We see that the binocular neurons do need to learn to the non-stimulated neurons.

3.3.2 Truncating negative weights to zero

Are negative weights necessary? Are weights of 0 sufficient?

```
for x = 1:N;
    for y = 1:N;
        if L(x,y) < 0;
            L(x,y) = 0;
        end
    end
end
```

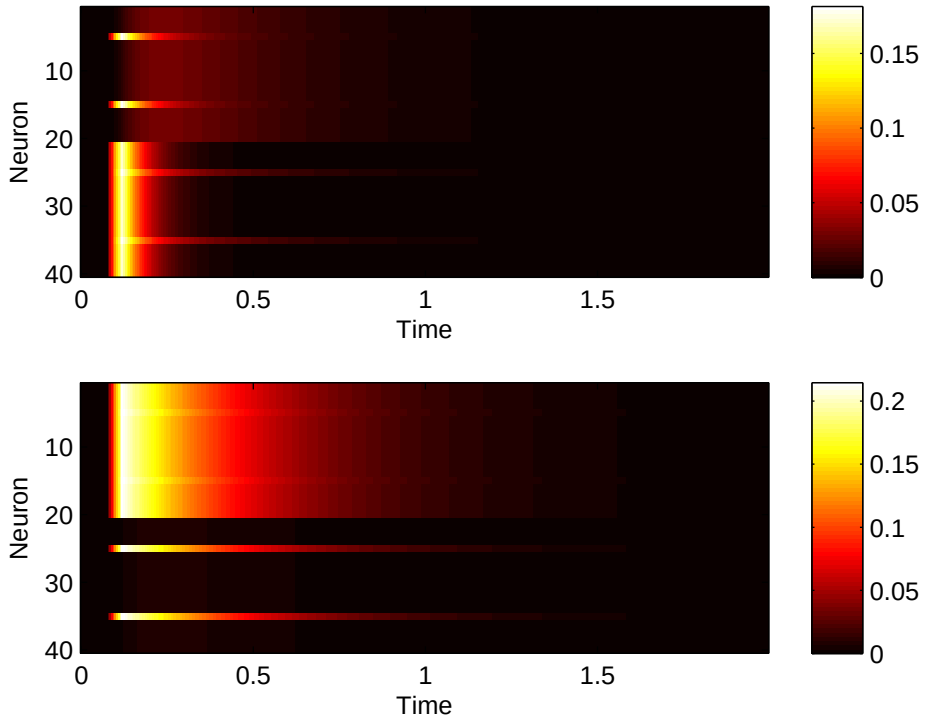


Figure 3.12: Neural activity after allowing only positive weights

Neuronal activity is similar to the original problem with binocular neurons. Eyes are not separated and one subset has not been trained to the correct time. We see that negative weights are necessary (or at the very least, the weights cannot be zero).

3.3.3 Subtracting small values from the cross subset weights

Rather than modifying the Hebbian learning rule to create negative cross subset weights, what if we did that manually?

```
L_p(1:20,21:40) = L_p(1:20,21:40) - .004;
L_p(21:40,1:20) = L_p(21:40,1:20) - .00025;
```

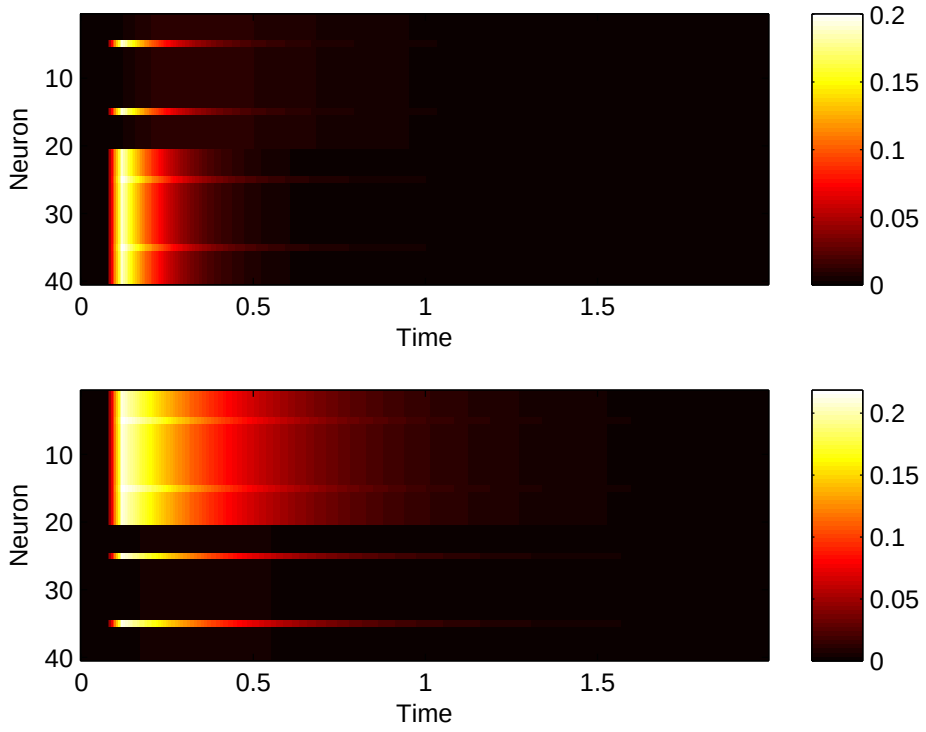


Figure 3.13: Neural activity after subtracting small amounts from cross subset weights

Separation is good and nearly complete. For the most part, timing is correct (subset 2 neurons are just shy of .5s).

3.4 Population activity across epochs

Binocular networks can still learn the correct times with corrected Hebbian functions.

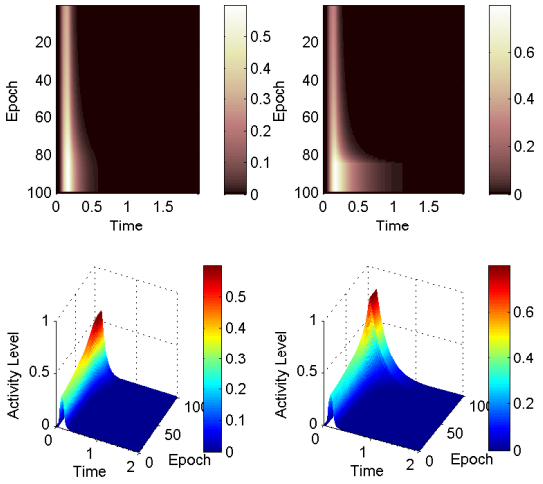


Figure 3.14: Monocular population activity (No binocular neurons)

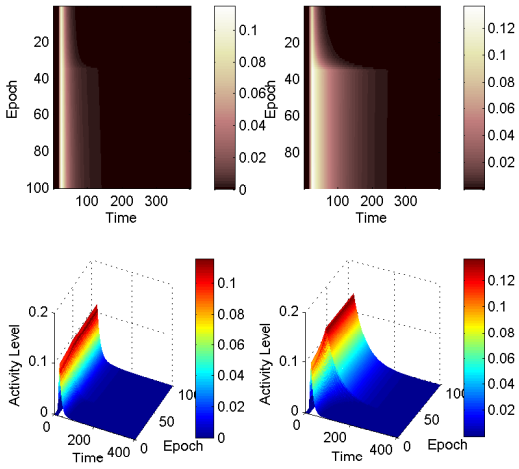


Figure 3.15: Binocular population activity (10% overlap)

Within each figure, each side represents a different neuron subset. The left side of each shows the subset trained to .5s while the other side is trained to 1s. Within the two figure sets, the pink colormapped image is in a (0, 90) azimuth/elevation viewing angle while the figure directly below each is in a viewing angle of (25, 30).

We find that with the corrected Hebbian functions (in the case of these plots, function **3**), network activities still grow to the correct time. (Learning parameters are different for the two sets, hence different learning rates, but the differential equations governing the learning mechanics are all the same.)

Chapter 4

Discussion

The core of this model is its ever-changing synaptic weights. It is on the assumption that these alone can correctly mimic Shuler and Bear's experimental results. Though the research is not extensive, there are experimental results that indicate It is then no surprise to find that most differences we see in neuronal activity across Hebbian forms can be correlated with differences in the weight matrices. We found some peculiarities which we currently do not have an explanation for, presented below.

As expected, weights connecting subset 1 to itself are stronger than any others (aside from a binocular neuron to itself) to allow for the slowest decay through positive feedback. With a shorter decay, subset 2 weights to itself are smaller. Due to our implementation method, binocular neurons are active every epoch. Thus, they form positive connections to all other neurons. In trying to completely separate the activity of the two neuronal subsets, we would expect the cross subset weights to be negative as to oppose the cross subset activation from the binocular neurons. In all correctly learning networks, we found these negative weights. Furthermore, if we set all entries in the final weight matrices to zero and run another epoch, we see visible activation in the non-stimulated neurons (figure **17**). Thus, we find it necessary (though not compulsorily sufficient) for the cross subset weights to be negative.

In all scenarios that produced a corrected network, corrections were done asymmetrically - either only the postsynaptic or presynaptic activity was corrected. When both are treated equally, as in Hebbian function **4**, we receive unwanted results. Yet in the original model as reported by Gavornik et al, a simple correction factor of subtracting 2.5 from each **V** was sufficient to resolve the two eyes. There is likely to be some sort of bug in our code. However, comparison of the two models so far has not revealed significant differences that would result in this discrepancy.

While on the subject of Hebbian function **4**, we come to another finding we cannot explain. Despite the two subsets decaying to essentially the same time, the within subset weights are profoundly different (one is nearly double the other). Yet one of the bases of our models is that the larger the weights, the longer the decay. We find this irregularity only in the case of function **4**.

If we assume that the activity of a stimulated neuron never decreases below 0 , working network corrections effectively impose a potentiation threshold upon the postsynaptic neuron where below that point the change in the protoweights will not be positive (these thresholds are 0 in network **1** and θ_1 in networks **2** and **3**). This does not preclude synaptic depression however, as the presynaptic neuron is subjected to a threshold that allows for values to decrease below 0 (i.e. if V is less than θ_1).

On this note, our findings also diverge a bit from those originally reported by Gavornik *et al.* While the original model "does not include a substantive role for inhibition in the creation of temporal representations[10]," our results indicate that the negative weights, i.e. inhibition, are necessary. Review of work on pyramidal cell networks by Barbour *et al.* seems to indicate that inhibitory signals in a recurrent, attractor network are unnecessary, but zero weights are[11] - on which both accounts our model digresses. However, our model is only a pseudo-attractor network and so those findings are not fully applicable. But if we were to consider attractor networks, our model does exhibit the delay

periods (inhibited firing until the end of stimulation) found to be necessary in a long-term synaptically plastic network by Amit and Brunel[12]. Also, the biological existence of inhibitory circuit plasticity would seem to suggest they are actively involved in learning[13,14].

Our Hebbian functions changes result in inhibitory weights for synapses between neurons in opposing subsets. How exactly then, do these corrections promote the formation of negative weights between the correct neurons? If we take function 1, for example, and run a subset 2 simulation, we find the presynaptic activity quantity $(\mathbf{V} - \theta_1)$ is negative for neurons 1-4, 6-14, 16-20 and positive for the rest. The postsynaptic activity quantity is positive for all neurons. When we take the product of these two quantities to calculate change in the protoweights, we find a decrease from any connections made from subset 1. We can apply this same reasoning in reverse when we stimulate subset 1 instead of 2. On the other hand, if we subtracted θ_1 from the presynaptic neuron, the presynaptic quantity for neurons 1-4, 6-14, and 16-20 would be 0 rather than negative and there would be no inhibition. Binocular neurons would increase the activity of all neurons and thus we would not get separation.

The constraint $(\mathbf{V} - \theta_1) > 0$ also performs a similar transformation as $\mathbf{(V)}$. If $\mathbf{V}$ is less than θ_1, the activity quantity for that neuron will be set to 0 and result in no change in protoweights. Thus, networks 2 and 3 work in a somewhat similar fashion as network **1**.

If our understanding of how these functions work is correct, it would imply that these functions are in and of themselves not unique and are merely a means to reach an asymmetric weight matrix with negative cross subset weights. Indeed, this does seem to be the case as it is possible to reach a correctly trained network with Hebbian function simply V*V but manually subtracting small values from quadrants 1 and 3 of the weight matrix. While this method of directly depressing synaptic weights to allow for both inhibitory and excitatory synapses from the same neuron is biologically infeasible, it opens the possibility of creating more precise models since we have elucidated the conditions necessary for a working model.

Chapter 5

Annotations

[3]Mauk M.D. & Buonomano D.V. 2004. The neural basis of temporal processing. *Annual Review of Neuroscience*. 27: 307-40.

[4]Buonomano D.V. & Merzenich M.M. 1995. Temporal information transformed into a spatial code by neural network with realistic properties. *Science*. 267: 1028 - 1030.

[5]Deisz R.A. & Prince D.A. 1989. Frequency-dependent depression of inhibition in guinea-pig neocortex in vitro by $GABA_B$ receptor feed-back on GABA release. *Journal of Physiology*. 412: 513 - 541.

[6]Newberry N.R. & Nicoll N.A. 1984. A bicuculline-resistant inhibitory post-synaptic potential in rat hippocampal pyramidal cells in vitro. *Journal of Physiology*. 348: 239 - 254.

[7]Shuler M. G. & Bear M.F. 2006. Reward timing in the primary visual cortex. *Science*. 311:1606 1609

[8]Funahashi S., Bruce C.J., Goldman-Rakic P.S. 1989. Mnemonic coding of visual space in the monkey's dorsolateral prefrontal cortex. *J. Neurophysiol*. 61: 331 349.

[9]Nakamura K. & Kubota K. 1995. Mnemonic firing of neurons in the monkey temporal pole during a visual recognition memory task. *J. Neurophysiol*. 74: 162 178. [10]Gavornik, J.P. et al. 2009. Learning reward timing in cortex through reward dependent expression of synaptic plasticity. *Proceedings of the National Academy of Science*. 106 (16): 6826-6831.

[11]Barbour B., Brunel N., Hakim V. & Nadal J.P. 2007. What can we learn from synaptic weight distributions? *Trends in Neuroscience*. 30 (12): 622 - 629.

[12]Amit D.J. & Brunel N. 1997. Model of global spontaneous activity and local structured activity during delay periods in the cerebral cortex . *Cerebral Cortex*. 7: 237 - 252.

[13]Pelkey K.A., Lavezzari G., Racca C., Roche K.W., & McBain C.J. 2005. mglur7 is a metaplastic switch controlling bidirectional plasticity of feedforward inhibition. *Neuron*n. 46: 89102.

[14]Soler-Llavina G.J. & Sabatini B.L. .2006. Synapse-specific plasticity and compartmentalized signaling in cerebellar stellate cells. *Nature Neuroscience*. 9: 798806.

Chapter 6

Acknowledgements

Special thanks to:

- Harel Shouval for guidance and mentorship,
- Naveed Aslam for encouragement and consultation,
- and the Gulf Coast Consortia, particularly Steve Cox, and the National Science Foundations for program sponsorship.

This work was partially funded by NSF REU Grant DMS-0755294

Chapter 7

Appendices - Matlab programs

7.1 network86
neuron network, across epochs, binocular stimulation

The following Matlab program serves as the basis of calculating a network of neuron's activity over the course of several hundred epochs. Neurons are presented with a reward signal and are able to learn. Stimulation can be either binocular or monocular. The different Hebbian functions were tested using this program.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Simulates activity of a recurrent, learning network of neurons             %%
%% over time when given an input stimulus and reward                          %%
%%                                                                            %%
%% version 1: one neuron feeding back on to itself                            %%
%% version 2: attempt at more than one neuron + 3-d plot                      %%
%% version 3: 3-d graphing                                                    %%
%% version 4: attempt to correct 3-d graphing + add learning rule             %%
%% errors in version 4 - Computation of V often produces either Inf or NaN,   %%
%% especially when user-defined variables are changed, particularly N.        %%
%% version 5: randomization of most parameters, changed format to cell mode.  %%
%% also calculating two networks, each with a different reward time           %%
%% version 6: cleaning up of code - combining networks in to one large loop   %%
%% version 8: add option for discontinuous plotting using plot3               %%
%% version 86: binocular inputs                                               %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Parameters
clf; % clear figure
dt = 0.005; % step in time
T = 2; % length of run time
epochs = 100; % number of epochs network activity simulated over
N = 40; % number of neurons
tau_m = .06; % time constant
tau_p = 5; % proto-time constant
eta = 2.2; % learning rate
r = .3; % reward magnitude
beta = 50; % sets activity level when reward is fully inhibited
rt1 = .5;
rt2 = 1;
rt = 50000;
```

```
theta1 = 0.0;
theta2 = 0.0;
theta3 = 0.0;

%% Matrices and vectors
ln_T = T/dt;
t = 0:dt:T-dt;
t_e = 1:epochs/2;
n = 1:N;
L = zeros(N,N); % initialize learning network weights
I_1 = zeros(N,1);
I_2 = zeros(N,1);
V = zeros(N,ln_T); % initialize learning network activity to be zero
% V_3 = zeros(epochs/2, ln_T-1);
% V_4 = zeros(epochs/2-1, ln_T-1);
I = zeros(N,1); % creating matrix space for input
for y = N/2+1:N;
    I_1(y) = .3;
end

for y = 1:N/2;
    I_2(y) = .3;
end

% I_1(5) = .3;
% I_1(15) = .3;
% I_2(25) = .3;
% I_2(35) = .3;

%% Activity loops
for j = 1:(epochs-1);
    j
    L_p = zeros(N,N);
    for i = 1:(ln_T-1);
        if i > 15 && i < 25;
            if mod(j,2) == 0;
                I = I_2;
                rt = rt2/dt;
            else
                I = I_1;
                rt = rt1/dt;
            end
        else I(:) = 0;
        end
        if i > rt-5 && i < rt+5;
            L = L + dt * eta * L_p(:,:) * (r - beta * mean(V(:,i)));
        end

%         V_5(j,i) = mean(V(:,i));

%         if mod(j,2) == 1;
%             V_3((j+1)/2,i) = mean(V(:,i));
%         else
%             V_4((j)/2,i) = mean(V(:,i));
```

```
%         end
        V(:,i+1) = V(:,i) + (dt/tau_m) * (-V(:,i) + I + L*V(:,i));
    L_p(:,:) = L_p(:,:) + (dt/tau_p) * (-L_p(:,:) + ((V-theta3)*(V'-theta1)) - theta2);
%   L_p(1:20,21:40) = L_p(1:20,21:40) - .00004;
%   L_p(21:40,1:20) = L_p(21:40,1:20) - .000025;
%   L_p(1:20,21:40) = 0;
%   L_p(21:40,1:20) = 0;
    end

    if j == 99;
        V_1 = V;
    end
    if j == 98;
        V_2 = V;
    end

%     if j >= 99;
%         for x = 1:N;
%             for y = 1:N;
%                 if L(x,y) < 0;
%                     L(x,y) = 0;
%                 end
%             end
%         end
%     end

%     if j == 100;
%         V_3 = V;
%     end
%     if j == 101;
%         V_4 = V;
%     end
end

%%
subplot(2,1,1);
imagesc(t,n,V_1)
colormap hot
colorbar

%%
subplot(2,1,2);
imagesc(t,n,V_2)
colormap hot
colorbar

%%
figure
imagesc(L)
colorbar

%%
% figure
% subplot(2,1,2);
```

```
% imagesc(t,n,V_3)
% colormap hot
% colorbar
% subplot(2,1,1);
% imagesc(t,n,V_4)
% colormap hot
% colorbar
```