Plasticity

A Synaptic Modification
Simulation Environment
http://web.bryant.edu/~bblais/plasticity

Author: Brian S. Blais

bblais@bryant.edu

March 16, 2004

Documentation version 0.7

Contents

Chapter 1

Introduction

There is very little documentation for this program (what you see here is basically all there is).

On the plus side, most of the parts of this program are really self-explanatory, so there isn't much need. I will outline the basic parts of the program here, and you can always send me questions by email. What may happen is that I make the email support files into the documentation for the program.

1.1 What is Plasticity?

Plasticity is a package of programs, with a convenient interface, used to run simulations of single cells and networks of neurons. The basic principle of the simulations is the presentation of pattern and noise vectors to a network of neurons, with specified lateral connectivity and synaptic modification rule. The pattern vectors are either patches from images or simply read from a Matlab .mat file as columns of a matrix. The noise vectors are generated during run-time. The interface lets you specify many things including

- the dimensionality of the inputs
- number of channels (left/right eye, ON/OFF channels, etc.)
- the pattern input files
- generation of input files from a specified correlation function
- the noise mean, variance and type (uniform, Gaussian, etc.)
- the lateral connectivity
- the learning rule
- constraints on the weights (normalization, saturation limits, etc.)
- number of neurons
- number of iterations
- what information to display

1.2 What do I do first?

When you first run 'plasticity', it is set up to run a single neuron, with two channels (left and right eye), with pattern input taken as patches from natural scenes and the BCM modification rule. After starting the interface, you should be able to hit "Go!" and the simulation will run and you will see the oriented receptive fields form, the modification threshold converge, and the orientation tuning plots become selective.

You can start and stop the simulation many times. You can do to the "Images" menu under the "Input Environment" menu (same as the Input button) and choose from a number of standard experiments (monocular deprivation = noise in one channel, pattern in another; binocular deprivation = noise in both channels; etc.). You can also click on an image for a channel and change what the inputs are for that channel. This way you can make an ON/OFF channel model, by changing the inputs of one of the channels to an image with the OFF channel properties. You can click on the noise image to change the properties of the noise into that channel. If both pattern and noise are specified, they are simply added.

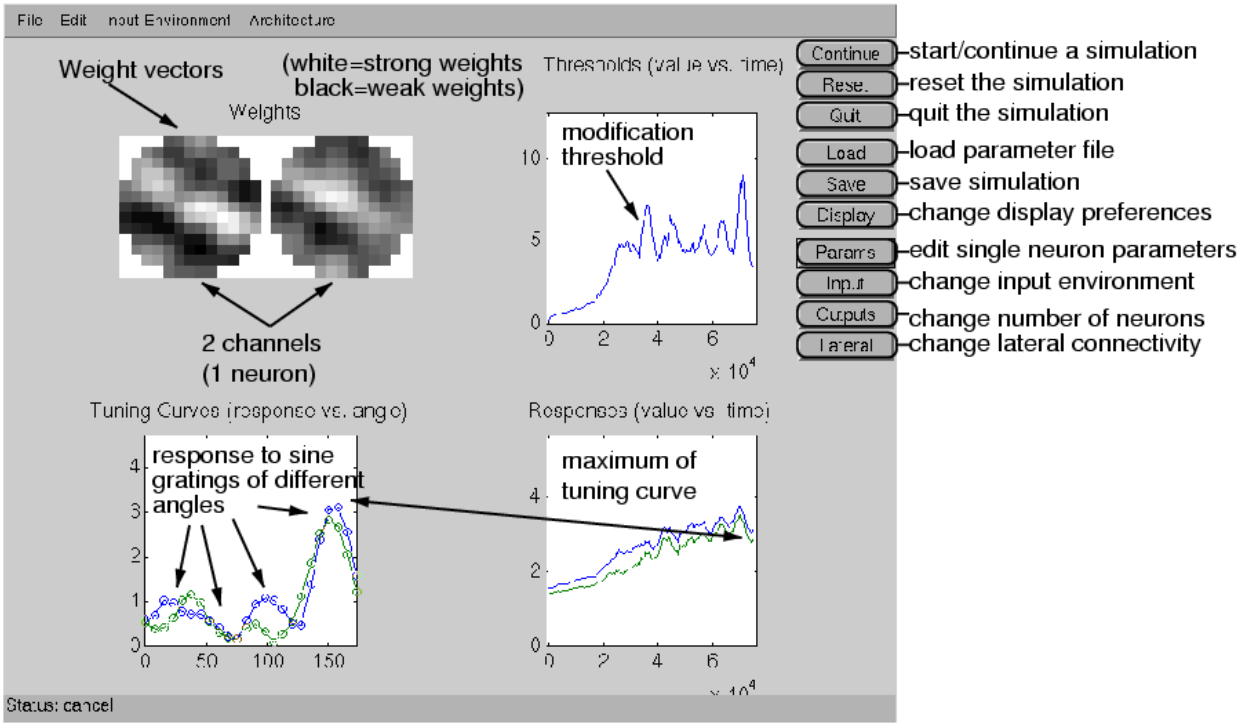
You can change the learning rule by going to the "Single Neuron Parameters" menu under the "Edit menu" (same as the Params button). There are number of built-in learning rules and there is currently no way to add your own without going into the code. I tried to make it more general, but the loss of speed (over 100x!) made me not include that option. If you have a learning rule, you can email me (bblais@bryant.edu) and I can put it in and send you an updated binary (or source code, if you're compiling it yourself).

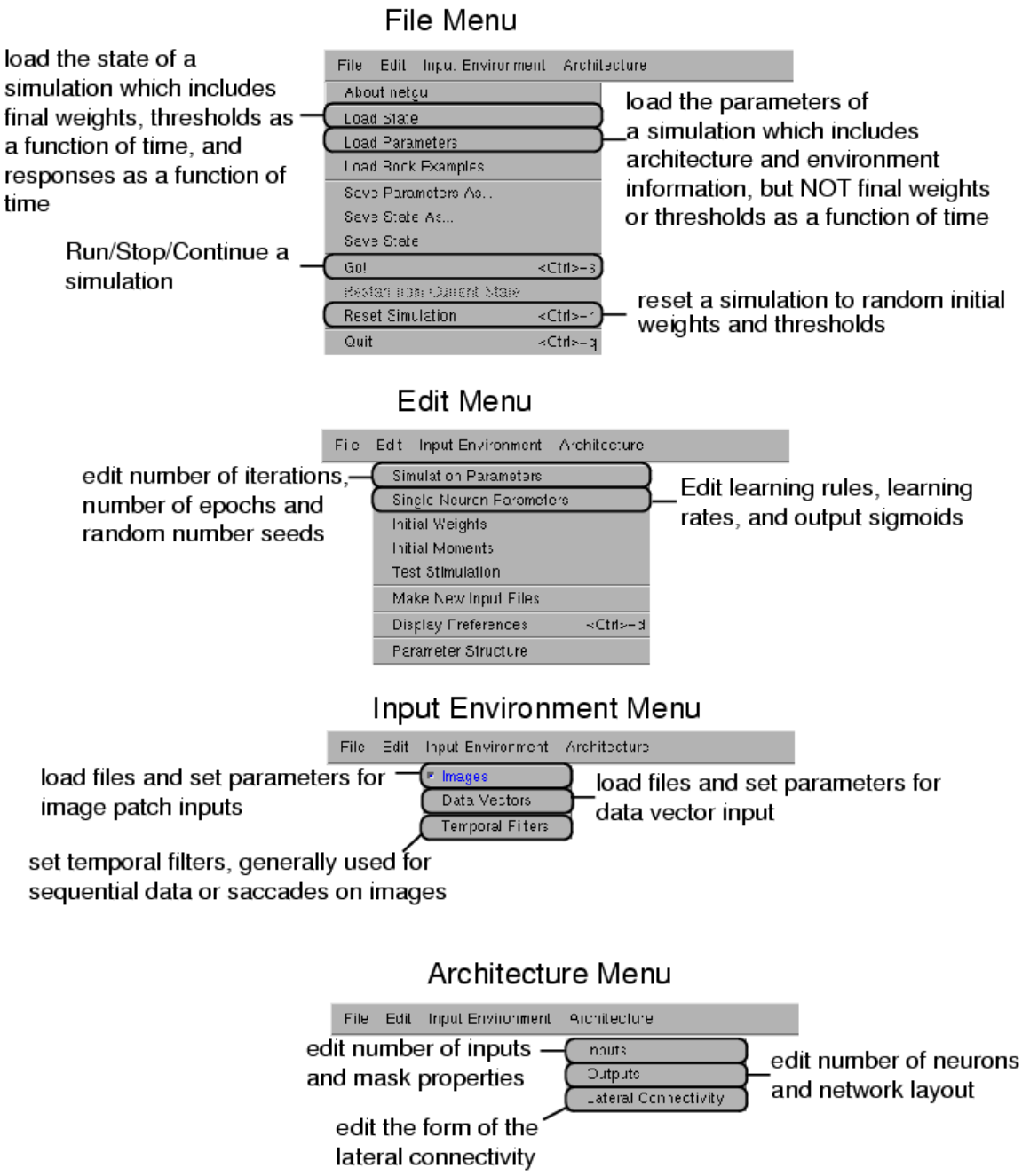
Chapter 2

Menus

2.1 Screenshots

Here are a couple of screen shots, with short explanations of the menus.





2.2 File

Load State

Load the state of a simulation. The state includes the simulation parameters (defined below), the values of the final weights and moments, and the time development of the moments and test stimulus results. Everything which defines a simulation is contained in the state.

Load Parameters

Load the parameters of a simulation. The parameters contain enough information to run a simulation, but *does not include* the results of a simulation. The parameters include the names of the input files, the definitions of the learning parameters, architecture, number of iterations, the initial conditions, etc.

Load Book Examples

There are a number of standard simulations which are presented in the book "Theory of Cortical Plasticity". All of the figures from the book which come from simulations are saved here, as simulation files. You can reproduce the results of any of the figures.

Save Parameters As...

Save the parameters of a simulation (as opposed to the state). This saves enough information to run the current simulation, but does not contain the results of the current simulation.

Save State As..., Save State

Save the state of a simulation, which includes a full definition of the simulation and the current results of the simulation.

Go! (Stop, Continue)

Selecting "Go!" runs a simulation with the current parameters. One can then select "Stop" to stop the simulation mid-way. Selecting "Continue" after this will continue the simulation. You can change any of the parameters of a simulation before continuing the simulation, as long as the architecture doesn't change. If the architecture changes (number of channels, inputs per channel, or number of neurons) then it doesn't make sense to continue the simulation.

Restart from Current State

Restart the simulation by setting the initial conditions to the current conditions.

Reset Simulation

Reset the simulation by resetting the initial conditions to default (and usually random) conditions. The parameters are not modified by this.

Page Setup

Sets up the page for printing.

Print

Prints the page.

Quit

Quit the simulation. If the simulation has been modified, then it will ask you whether you want to save it. It will save the state in this case, not just the parameters, as defined above.

2.3 Edit

Simulation Parameters

Random Number Generator

Usually the random number generator is seeded with the clock. This way, two simulations starting with the same parameters will have different initial conditions and sequence of input patterns and noise. You can specify the seed by hand, if you want to completely reproduce exactly as it was run.

Number of iterations

A simulation runs for a certain number of iterations. This is broken up into epochs. After each epoch the display is updated. Quantities such as thresholds and responses to test stimulus are stored every epoch, not every iteration. Displayed is a rough outline of the program with an inner loop and outer loop to make this clear. The inner loop determines how many iterations per epoch, while the outer loop determines how many epochs to run. Of course the number of epochs actually run depends on whether you start and stop the simulation by hand.

Save weights at every epoch

Determines if the weights are saved at every iteration. This can make for some very large files, so the default setting is Off. For low dimensional cases, or if the epoch number is low, one may find this feature useful but for the most part it is not necessary.

Use Single Precision

By default, Matlab uses double precision numbers for all calculations. You can make the simulations calculate most numbers in single precision to save memory. This is particularly useful when dealing with large networks.

Save Lateral Matrix

By default, if a lateral connection matrix is specified by a file, then that connection matrix is not included in the state file. You can have it saved with the state file with this option.

Epochs per Display

If you are running the program over a network, and it is a slow connections, displaying every epoch is not practical. Setting this option makes Plasticity still calculate and save every epoch, but only updates the display after a certain number of epochs has passed.

Single Neuron Parameters

Learning Rules

This menu allows you to select from a wide array of learning rules. Relevant citations are given below, as well as a few online references.

The references are not meant to be complete, but merely used as a starting point.

- **BCM**

  See also Law94 BCM, Intrator92 BCM, and BCM Up/Down below.

  Bienenstock, E. L., Cooper, L. N., and Munro, P. W. (1982). Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, 2:32–48.

  Clothiaux, E. E., N Cooper, L., and Bear, M. F. (1991). Synaptic plasticity in visual cortex: Comparison of theory with experiment. *Journal of Neurophysiology*, 66:1785–1804.

  Blais, B., Cooper, L. N., and Shouval, H. (2000). Formation of direction selectivity in natural scene environments. *Neural Computation*, 12(5).

  Blais, B., Shouval, H., and Cooper, L. N. (1999). The role of presynaptic activity in monocular deprivation: Comparison of homosynaptic and heterosynaptic mechanisms. *Proc. Natl. Acad. Sci.*, 96:1083–1087.

- **Law94 BCM**

  Law, C. and Cooper, L. (1994). Formation of receptive fields according to the BCM theory in realistic visual environments. *Proceedings National Academy of Sciences*, 91:7797–7801.

  Shouval, H., Intrator, N., Law, C. C., and Cooper, L. N. (1996b). Effect of binocular cortical misalignment on ocular dominance and orientation selectivity. *Neural Computation*, 8(5):1021–1040.

  Shouval, H., Intrator, N., and Cooper, L. N. (1997a). BCM network develops orientation selectivity and ocular dominance in natural scene environment. *Vision Research*, 37:3339–3342.

- **Hebb**

  See also Hebb No DC and Linsker below.

  Hebb, D. O. (1949). *The Organization of Behavior; a neuropsychological theory.* Wiley, New York.

  Oja, E. (1982). A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15:267–273.

Shouval, H., and Cooper, L. N (1996) Organization of receptive fields for networks of neurons with hebbian type learning: The connection between synaptic and phenomenological models. *Biological Cybernetics*, 73: 439–447.

- **Intrator92 BCM**

  Intrator, N. and Cooper, L. N. (1992). Objective function formulation of the BCM theory of visual cortical plasticity: Statistical connections, stability conditions. *Neural Networks*, 5:3–17.

  Blais, B., Intrator, N., Shouval, H. and Cooper, L. N. (1998). Receptive field formation in natural scene environments: comparison of single cell learning rules *Neural Computation*, 10(7).

  Blais, B., Shouval, H., and Cooper, L. N. (1999). The role of presynaptic activity in monocular deprivation: Comparison of homosynaptic and heterosynaptic mechanisms. *Proc. Natl. Acad. Sci.*, 96:1083–1087.

- **Kurtosis 1**

  Blais, B., Intrator, N., Shouval, H. and Cooper, L. N. (1998). Receptive field formation in natural scene environments: comparison of single cell learning rules *Neural Computation*, 10(7).

  Blais, B., Shouval, H., and Cooper, L. N. (1999). The role of presynaptic activity in monocular deprivation: Comparison of homosynaptic and heterosynaptic mechanisms. *Proc. Natl. Acad. Sci.*, 96:1083–1087.

  Blais, B., Cooper, L. N., and Shouval, H. (2000). Formation of direction selectivity in natural scene environments. *Neural Computation*, 12(5).

- **Kurtosis 2**

  Hyvarinen, A. and Oja, E. (1996). A fast fixed-point algorithm for independent component analysis. *Int. Journal of Neural Systems*, 7(6):671–687.

  Blais, B., Intrator, N., Shouval, H. and Cooper, L. N. (1998). Receptive field formation in natural scene environments: comparison of single cell learning rules *Neural Computation*, 10(7).

  Blais, B., Shouval, H., and Cooper, L. N. (1999). The role of presynaptic activity in monocular deprivation: Comparison of homosynaptic and heterosynaptic mechanisms. *Proc. Natl. Acad. Sci.*, 96:1083–1087.

  Blais, B., Cooper, L. N., and Shouval, H. (2000). Formation of direction selectivity in natural scene environments. *Neural Computation*, 12(5).

- **Skewness 1**

  Blais, B., Intrator, N., Shouval, H. and Cooper, L. N. (1998). Receptive field formation in natural scene environments: comparison of single cell learning rules *Neural Computation*, 10(7).

  Blais, B., Shouval, H., and Cooper, L. N. (1999). The role of presynaptic activity in monocular deprivation: Comparison of homosynaptic and heterosynaptic mechanisms. *Proc. Natl. Acad. Sci.*, 96:1083–1087.

Blais, B., Cooper, L. N., and Shouval, H. (2000). Formation of direction selectivity in natural scene environments. *Neural Computation*, 12(5).

- **Skewness 2**

  Blais, B., Intrator, N., Shouval, H. and Cooper, L. N. (1998). Receptive field formation in natural scene environments: comparison of single cell learning rules *Neural Computation*, 10(7).

  Blais, B., Shouval, H., and Cooper, L. N. (1999). The role of presynaptic activity in monocular deprivation: Comparison of homosynaptic and heterosynaptic mechanisms. *Proc. Natl. Acad. Sci.*, 96:1083–1087.

  Blais, B., Cooper, L. N., and Shouval, H. (2000). Formation of direction selectivity in natural scene environments. *Neural Computation*, 12(5).

- **BCM Up/Down**

  No references.

- **Hebb no DC**

  Miller, K. D. (1994). A model for the development of simple cell receptive fields and the ordered arrangement of orientation columns through activity-dependent competition between on- and off-center inputs. *J. Neurosci.*, 14:409–441.

  MacKay, D. J. C. and Miller, K. D. (1994). The role of constraints in Hebbian learning. *Neural Computation*, 6:100–126.

  Erwin, E. and Miller, K. D. (1998). Correlation-based development of ocularly matched orientation and ocular dominance maps: determination of required input activities. *J Neurosci.*, 18(23):9870–95.

  Erwin, E., Obermayer, K., and Schulten, K. (1995). Models of orientation and ocular dominance in visual cortex. *Neural Computation*, 7.3:425–468.

- **Linsker**

  Linsker, R. (1986a). From basic network principles to neural architecture: emergence of orientation columns. *PNAS*, 83:8779–8783.

The following lists the equations for the included learning rules. The notation differs slightly from the notation in "The Theory of Cortical Plasticity". It follows more closely the notation in my PhD Dissertation.

Notation	
What is it?	Symbol
Input from i-th input to j-th neuron	x_{ij}
Weight from i-th input to j-th neuron	w_{ij}
Output of j-th neuron	y_j
Modification Threshold An activity threshold marking a change from weight decreases to weight increases	θ_j^M
Scaling "threshold" A value, usually related to the modification threshold, used to scale the learning rate of a learning rule. The fixed points are not affected by this value, only the dynamics.	θ_j^S
Learning Rate	η
Memory Constant Used in calculating averages for thresholds (see definitions below)	τ
Lateral connection from k-th neuron to j-th neuron	L_{jk}
Sigmoid Used in calculating the outputs (see definitions below)	σ
Derivative of Sigmoid Used in some learning rules	σ'

Definitions	
What is it?	Symbol
Average of n-th power of the output	$E_\tau\left[y_j^n\right] = \frac{1}{\tau}\int_{-\infty}^{t} y^n(t')e^{-(t-t')/\tau}dt'$
Differential Equation for Average of n-th power of the output	$\frac{d\xi}{dt} = \frac{1}{\tau}\left(y^n - E_\tau\left[y_j^n\right]\right)$
Output of cell without Lateral Connectivity	$y_j = \sigma\left(\sum_i w_{ij}x_{ij}\right)$
Output of cell with Lateral Connectivity	$y_j = \sigma\left(\sum_i w_{ij}x_{ij} + \sum_k L_{jk}\sigma(\sum_i w_{ik}x_{ik})\right)$

Learning Rule Equations

BCM

$$\begin{aligned} \Delta w_{ij} &= \eta y_j\left(y_j - \theta_j^M\right)x_{ij} \\ \theta_j^M &= E[y_j^2] \end{aligned}$$

Law94 BCM

$$\begin{aligned} \Delta w_{ij} &= \eta y_j\left(y_j - \theta_j^M\right)x_{ij}/\theta_j^S \\ \theta_j^M &= E[y_j^2] \end{aligned}$$

$$\theta_j^S = E[y_j^2]$$

Hebb

$$\Delta w_{ij} = \eta y_j x_{ij}$$

Intrator92 BCM

$$\Delta w_{ij} = \eta y_j \left(y_j - \theta_j^M\right) x_{ij} \sigma_j'$$
$$\theta_j^M = E[y_j^2]$$

Kurtosis 1

$$\Delta w_{ij} = \eta y_j \left(y_j^2 - \theta_j^M\right) x_{ij} \sigma_j' / \theta_j^S$$
$$\theta_j^M = E[y_j^4]/E[y_j^2]$$
$$\theta_j^S = E^2[y_j^2]$$

Kurtosis 2

$$\Delta w_{ij} = \eta y_j \left(y_j^2 - \theta_j^M\right) x_{ij} \sigma_j'$$
$$\theta_j^M = 3E[y_j^2]$$

Skewness 1

$$\Delta w_{ij} = \eta y_j \left(y_j - \theta_j^M\right) x_{ij} \sigma_j' / \theta_j^S$$
$$\theta_j^M = E[y_j^3]/E[y_j^2]$$
$$\theta_j^S = E^{1.5}[y_j^2]$$

Skewness 2

$$\Delta w_{ij} = \eta y_j \left(y_j - \theta_j^M\right) x_{ij} \sigma_j'$$
$$\theta_j^M = E^{0.5}[y_j^2]$$

BCM Up/Down

$$\Delta w_{ij} = \eta y_j \left(y_j - \theta_j^M\right) x_{ij}$$
$$\theta_j^M = \begin{cases} E_{\tau_{\text{up}}}[y_j^2] & \text{if } y_j > \theta_j^M \\ E_{\tau_{\text{down}}}[y_j^2] & \text{if } y_j < \theta_j^M \end{cases}$$
$$\eta = \begin{cases} \eta_{\text{up}} & \text{if } y_j > \theta_j^M \\ \eta_{\text{down}} & \text{if } y_j < \theta_j^M \end{cases}$$

Hebb no DC

$$\Delta w_{ij} = \eta y_j x_{ij}$$

$$w_{ij} \quad \rightarrow \quad w_{ij} - \sum_{i=1}^{N_{\text{inputs}}} \Delta w_{ij}/N_{\text{inputs}}$$

Linsker

$$\Delta w_{ij} \quad = \quad \eta(y_j - y_o)(x_{ij} - x_o)$$

Stabilization Warning

A text message states whether a learning rule is stable without any extra stabilization (such as weight normalization or saturation limits). This does not include instabilities resulting from improper parameters. It represents a general property of the learning rule. For example, Hebbian learning is intrinsically unstable and requires some form of weight decrease or limit. BCM learning, on the other hand, is intrinsically stable and does not require such stabilization.

Extra Stabilization

Oja Normalization This is a local form of normalization given by Oja, 1982. It uses a decay term where the only information a synapse needs is its own value and the information needed generally for Hebbian learning. If the normal modification has the form

$$\Delta w_{ij} \quad = \quad \eta\phi(y_j)x_{ij}$$

then the Oja Normalized equations have the form

$$\Delta w_{ij} \quad = \quad \eta\phi(y_j)x_{ij} - \eta y_j\phi(y_j)w_{ij}$$

Strict Normalization After modification, the weights are updated according to the following:

$$w_{ij} \quad \rightarrow \quad \frac{w_{ij}}{\sum_i w_{ij}^2}$$

Saturation After modification, the weights above a given upper limit are set to the value of the upper limit, and weights below a given lower limit are set to the value of the lower limit.

Weight Decay The modification equation above is modified to include a constant decay value times the weight, as in the following:

$$\Delta w_{ij} \quad = \quad \eta\phi(y_j)x_{ij} - \eta\lambda w_{ij}$$

Learning Rate

The factor in front of every learning rule to determine how much the weights modify each iteration. (see Notation and Definitions above)

Memory Constant

The factor entering into all temporal average values calculated for thresholds. (see Notation and Definitions above)

Sigmoid

Determines how the output depends on the input and weights. Without lateral connectivity the output depends on the sigmoid like the following:

$$y_j = \sigma\left(\sum_i w_{ij}x_{ij}\right)$$

The forms of the sigmoid are the following.

None

$$\sigma(z) \;=\; z$$

Tanh

$$\sigma(z) \;=\; \begin{cases} z_{\text{top}}\tanh\left(\frac{z}{z_{\text{top}}}\right) & \text{if } z \geq 0 \\ z_{\text{bottom}}\tanh\left(\frac{z}{z_{\text{bottom}}}\right) & \text{if } z < 0 \end{cases}$$

Polynomial

$$\sigma(z) \;=\; z^{\text{exponent}}$$

Piecewise Linear

$$\sigma(z) \;=\; \begin{cases} z_{\text{top}} & \text{if } z \geq z_{\text{top}} \\ z & \text{if } z_{\text{bottom}} < z < z_{\text{top}} \\ z_{\text{bottom}} & \text{if } z \leq z_{\text{bottom}} \end{cases}$$

Initial Weights, Initial Moments

The range of the random initial weights and moments can be set, or the initial weights and moments can be loaded from a file.

Test Stimulation

The neurons are tested with sine gratings at different orientations to obtain the tuning curves. The spatial frequency and number of orientations can be set. The testing does not optimize over spatial frequency. It does optimize over phase. The default spatial frequency has been obtained empirically. Hebbian learning, Kurtosis and Skewness 2 all tend to give low spatial frequency receptive fields (except in whitened environments). For all other learning rules, the default is a higher spatial frequency test set.

The term *Channel* refers to groups of input cells. A two channel input could be two eyes, ON and OFF cells, or Lagged and Non-Lagged cells for example.

The term *Independent Channels* refers to testing each channel separately, as if all of the other channels are set to zero. *Summed Channels* refers to testing the whole neuron, summing the effects from all channels. The *Direction Selectivity Simple Case* refers to a special optimization that is only valid for a two channel system with one non-lagged channel and a single time-delay lagged channel.

Make New Input Files

The choices for making new input files are the following:

- **Pre-process Images**. Pre-process natural images, to normalize or apply filters such as difference of Gaussians or whitening.
- **Scatter Data**. Create 2D data sets, particularly clusters.
- **2x2 Correlation Function**. Create 2D data sets from a specified correlation function.
- **1xN Correlation Function**. Create N-D data sets from a specified correlation function.
- **Multi-Channel Correlation Function**. Create high-dimensional data sets from a specified correlation function, specifically representing multiple channels (such as ON-OFF, or left-right)

Start Making Movie

Start recording the frames of a movie, to be saved as an AVI or Matlab movie file. These files can get *very* large, so make sure to shrink the screen down to the smallest size still usable.

Display Preferences

Set the variables to be displayed, and their layout on the screen.

Parameter Structure

This menu is not intended for general use, and is included primarily for debugging purposes.

All of the parameters for the simulation are stored in a parameter structure called `params`. In the interface, this structure is kept free from errors and inconsistent data (for example, having a 2x2 connectivity matrix for 15 neurons). Editing this structure directly can lead to potential problems, so only edit this with care!

2.4 Input Environment

Images

Each channel is a pattern input file plus some type of noise. You can set the input images for the pattern input, and the distribution of noise (if any), by clicking on the pattern image or noise image.

Data Vectors

The inputs need not be patches from images, but can be any dimensional vectors. All a data file needs to have is a matrix with D rows, for the dimensionality of the input, and N columns, for the number of patterns. Not all of the drawing functions will work (or are appropriate) for data vectors.

Temporal Filters

The temporal filters change the way that inputs are processed per channel. For each channel, the value is

$$\mathbf{x} \;=\; \mathbf{x}(t)f(t) + \mathbf{x}(t-1)f(t-1) + \mathbf{x}(t-2)f(t-2) + \cdots$$

for all of the values of the filter f. This allows two channels to be delayed from each other (the filter for one having $f(t) = 0, f(t-1) = 1$ and the filter for the other having $f(t) = 1, f(t-1) = 0$).

2.5 Architecture

Inputs

This sets the size of the receptive field, and allows the circular mask to be removed.

Outputs

This sets the number of neurons, and the offset of their inputs. The default is that all neurons see exactly the same inputs.

Lateral Connectivity

The lateral connectivity can take many common forms, or be loaded from a file. The common forms are

- **1D Difference of Gaussians**. Short-range connections (in 1D) are excitatory, and medium range connections are inhibitory.
- **2D Difference of Gaussians**. Short-range connections (in 2D) are excitatory, and medium range connections are inhibitory.
- **Constant**. All neurons are connected to each other with constant value.
- **Restricted Constant**. All neurons within a certain radius are connected to each other with constant value.
- **Orthogonalization**. If you want to *force* cells in a network to find different solutions, then use one of the orthogonalization routines in the lateral connections menu. The standard orthogonalization works like this:

  1. The first neuron is not affected by anything, it will converge to the first solution. In the case of Hebbian learning, this will be the first principal component.
  2. The second neuron is forced to be orthogonal to the first, so it finds the next solution. In the case of Hebbian learning, this will be the second principal component.
  3. The third neuron is forced to be orthogonal to the first and second, etc.

  The later neurons will probably take longer to converge, and after a while (perhaps 20 or so neurons, depending on your environment) you won't have any convergence.

- **Symmetric Orthogonalization**.

  The symmetric orthogonalization forces all neurons to be orthogonal to each other all at once, so it's first come-first serve on the solutions. It might be the fifth neuron that converges to the first solution, and the third neuron to the second solution, etc. No neuron is special in this scenario.

2.6 Help

About Plasticity

Get some information about the Plasticity package, and a lovely picture drawn by the author himself.

Online Documentation

Access this document.

Chapter 3

How to add a new learning rule

In this section I am going to outline the steps to add a new learning rule. Although it is somewhat straightforward, there is a lot of room for error because you will be modifying the code directly. You should also make sure that you can compile the original files at all, because if you can't do that, then it will be impossible to add another learning rule.

3.1 Why is this so involved and difficult? Why not just a `.m` file?

It would be nice if life were easy, eh? It was suggested early on that a very nice feature would be to easily add a learning rule. All that would be required on my part to do this would be to insert into `train.c` a line which, if the learning rule doesn't exist, to use the name as an `.m` file. A simple call to `mexCallMATLAB` from there would easily solve the problem.

So I did this, realized immediately why it wasn't a good idea, and deleted it. Some time later someone else suggested it and, having forgotten about the previous time, I did it again. I realized again that it wouldn't work, and deleted it.

Why doesn't it work? It all comes down to one word: speed. The difference in speed between the current implementation and the `mexCallMATLAB` is well over a **factor of 200**! It essentially made the package useless. I felt that it was better not to have a feature than have a bad one, so we are left with the current method.

Even with the slow method, half of these directions would still have to be followed, because they are display issues not simulation issues.

3.2 The Example

In this example we will be adding two new learning rules: the Linsker Hebbian rule

$$\frac{d\mathbf{w}}{dt} = (\mathbf{x} - x_o)(y - y_o) \tag{3.2.1}$$

and a BCM learning rule with two thresholds.

There are two types of files that need to be changed. One deals with the interface aspect of the code, and the other deals with the actual calculations performed in the simulation.

The Interface Code

All of the interface code for new learning rules is contained in the `userdefined_learning_rules.m` file.

```
function p=userdefined_learning_rules(action,user_learning_rule,arg3);

  if (nargin==0)
    p=[];
    return;
  end

  if (strcmp(action,'default_params'))

    p(1).name='Linsker';
    p(1).value=[0;0];  % default values
    p(1).value_names='x_o|y_o';  % names of variables

    p(2).name='BCM Two Thresh';
    p(2).value=[1e5;4];  % default values
    p(2).value_names='tau2|th_max';  % names of variables

    return;

  end

  switch (user_learning_rule)

    case 1 % Linsker
      switch (action)
        case 'set_neuron_params_display'
          c=-2:.1:8;
          phi=4*c;
          plot(c,phi,'-');
          set(gca,'xtick',0,'ytick',0,'xlim',[-2 8],'ylim',[-10 24]);
          zeroaxes('k:');
          p=[.2 17.5]; p2=[.2 10]; p3=[.2 4];
          text(p(1),p(2),'dw/dt=(x-x_o)(y-y_o)','fontsize',12);

          %stability=1;  % stable
          stability=2;  % unstable
          %      stability=3;  % could be unstable
```

```
        p=stability;
      case 'get_num_moments'
        num_moments=1;

        p=num_moments;
      case 'get_threshold_from_moments'

        moments=arg3;
        p=moments(1,:,:);

      otherwise
        p=[];
    end
  case 2  % BCM with Two Thresholds

    switch (action)
      case 'set_neuron_params_display'
        c=-2:.1:8;
        phi=(c-1).*(c-5).*(c>=1);
        plot(c,phi,'-'); zeroaxes('k:');
        set(gca,'xtick',0,'ytick',0,'xlim',[-2 8],'ylim',[-10 24]);

        p=[.2 17.5]; p2=[.2 10]; p3=[.2 4];

        text(p(1),p(2),'\phi=(y-\theta_{2})(y-\theta_{1})/\theta_{1}^{2})','fontsize',12
        text(p2(1),p2(2),['\theta_{1}=E[y^2]']);
        text(p3(1),p3(2),['\theta_{2}=E_{slow}[y^2]']);

        stability=1;  % stable
        %      stability=2;  % unstable
        %      stability=3;  % could be unstable

        p=stability;
      case 'get_num_moments'
        num_moments=2;

        p=num_moments;
      case 'get_threshold_from_moments'

        moments=arg3;
        p=moments(1,:,:);

      otherwise
        p=[];
    end
end
```

The Simulation Code

The simulation code is broken up into two parts: the definition of the learning rule and the constraints applied after learning, `userdefined_learning_rules.c` and `userdefined_constraints.c` respectively.

A sample learning rules file is the following.

```
switch (user_learning_rule) {

case 1: /* Linsker */

  /* learning_rule_p[0] is the first userdefined parameter */
  /* learning_rule_p[1] is the second userdefined parameter */

  /* subtract the x_o parameter from the input vectors */

  for (ic=0,input_ptr=input; ic<num_channels; ic++) {
    for (j=0; (j<prod_num_neurons); j++) {
      for (i=0; i<inputs_per_channel;
           i++,input_ptr++)
          X-=learning_rule_p[0];
    }
  }

  /* calculate the moments */

  for (j=0,moments_ptr=moments_p,y_ptr=y_p;
       j<prod_num_neurons; j++,moments_ptr++,y_ptr++) {
    MOMENT0+=tau_inv*(Y*Y-MOMENT0);
  }


  /* calculate phi */
  for (j=0,moments_ptr=moments_p,phi_ptr=phi_p,y_ptr=y_p;
       j<prod_num_neurons; j++,moments_ptr++,y_ptr++,phi_ptr++)
      PHI=eta*(Y-learning_rule_p[1]);

  break;

case 2: /* BCM two thresh */
  /* learning_rule_p[0] is the first userdefined parameter */
  /* learning_rule_p[1] is the second userdefined parameter */

  /* calculate the moments */

  for (j=0,moments_ptr=moments_p,y_ptr=y_p;
```

```
       j<prod_num_neurons; j++,moments_ptr+=2,y_ptr++) {
    MOMENT0+=tau_inv*(Y*Y-MOMENT0);
    MOMENT1+=1.0/learning_rule_p[0]*(Y*Y-MOMENT1);
    if (MOMENT1>learning_rule_p[1])
        MOMENT1=learning_rule_p[1];
  }

  /* calculate phi */
  for (j=0,moments_ptr=moments_p,phi_ptr=phi_p,y_ptr=y_p;
       j<prod_num_neurons; j++,moments_ptr+=2,y_ptr++,phi_ptr++)
      PHI=eta*((Y-MOMENT1)*(Y-MOMENT0))*(Y>=MOMENT1);

  break;


}
```

Although not used for these rules, an example constraint file is

```
switch (user_learning_rule) {

case 1: /* rule with all positive weights */

  for (ic=0; ic<num_channels; ic++) {
    for (j=0; (j<prod_num_neurons);  j++) {

      w_idx=inputs_per_channel*j+inputs_per_channel*prod_num_neurons*ic;
      weights_ptr=&(weights_p[w_idx]);

      for (i=0; i<inputs_per_channel; i++,weights_ptr++)
          *weights_ptr=  ((*weights_ptr)<0) ? 0 : (*weights_ptr);


    } /* j */
  } /* ic */


  break;

}
```

Compile the code using `compile all` on the command line.

Appendix A

Installation

```
INSTALL Readme Version 0.7

Look at doc/plasticity_doc.pdf for more information.

Here are some directions for installing, and running, the plasticity simulation
package.  If you have any problems, it is entirely likely that it is my
fault.  :)  In that case, please email me (bblais@bryant.edu) with
questions.

#-------------------------------------------------------

Steps for Installing:

From the CDROM, on Windows:

1) Copy the entire plasticity folder from the CDROM to your harddrive

2) Make a shortcut to your Matlab application, and place it in the plasticity
folder

3) Right-click on the Matlab shortcut, and change the "Start In" directory to
be the full path of the plasticity folder.  For example, c:/mymatlab/plasticity

4) Start Matlab from the shortcut

5) type

      plasticity

   at the Matlab prompt

#-------------------------------------------------------
```

```
From the CDROM, on Unix:

1) Copy the entire plasticity directory from the CDROM to your harddrive

2) from within this directory, run matlab.

3) type

      plasticity

   at the Matlab prompt


#-------------------------------------------------------


From the web:

0) All of the directions hold from above, except that one uncompresses the
plasticity.tar.gz file in the directory of interest.  The decompression
procedure creates the plasticity directory.

****************CRUCIAL NOTE FOR WINZIP USERS*******************

When uncompressing this file, turn *off* the option to do "Smart CR/LF
Conversion" on TAR files (under Options/Configuration/Miscellaneous).  When
this option is on, Winzip messes up the .mat files, and causes Matlab to
crash rather violently.
************************************************************


If you are not on one of the supported architectures, then you will have to
compile the binaries yourself. You will need a C compiler, and have the proper
mexopts file made (consult the matlab documentation).  From the matlab prompt,
in the plasticity directory, execute the command

         compile all

       this should compile all of the mex functions for your particular
architecture.  if there are problems, you can contact me
(bblais@bryant.edu), although some problems may be specific to your
configuration.

NOTE: If you do manage to compile the files on an architecture I don't have,
```

please send me a note. I would like to include binaries from many different architectures to make the setup as easy as possible.

Brian Blais
bblais@bryant.edu