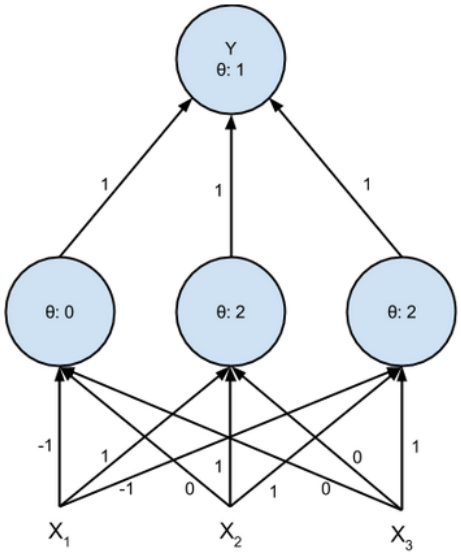
Alex Rosengarten
University of California, San Diego

January 29, 2015

HOMEWORK 1 — LaTeX

Problem 1. Logic with Perceptrons



(a)

(i) The inputs are X_1, X_2, X_3.

(ii) The activation for each neuron is specified by the following equation:

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x >= \theta \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

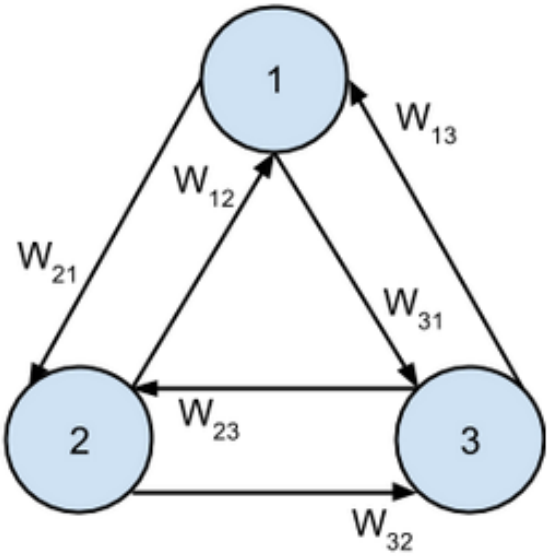
(iii) The output is Y.

(b)

X_1	X_2	X_3	H_1	H_2	H_3	Y
0	0	0	1	0	0	1
0	0	1	1	0	0	1
0	1	0	1	0	0	1
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	1	0	1
1	1	1	0	1	0	1

Problem 2. Hopfield network for strong patterns

(a)



The weight matrix for this network is as follows:

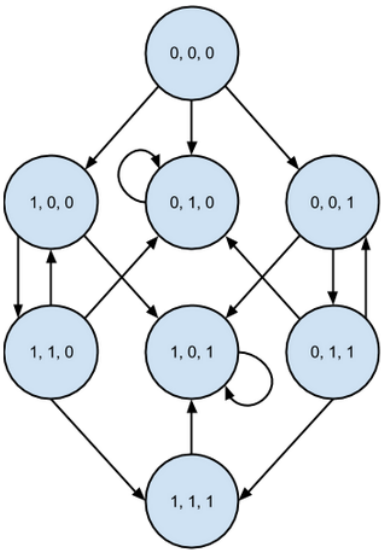
$$\begin{bmatrix} 0 & W_{12} & W_{13} \\ W_{21} & 0 & W_{23} \\ W_{31} & W_{21} & 0 \end{bmatrix} = \begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix}$$

Each weight was calculated with the following equation:

$$W_{ij} = \sum_{i=1}^{n} (2V_i - 1)(2V_j - 1) eq : hebbscalar \tag{2}$$

The vectorized version of the Hebbian learning rule is as follows:

$$W = W_0 + p * p^T - Ieq : hebbvec \tag{3}$$

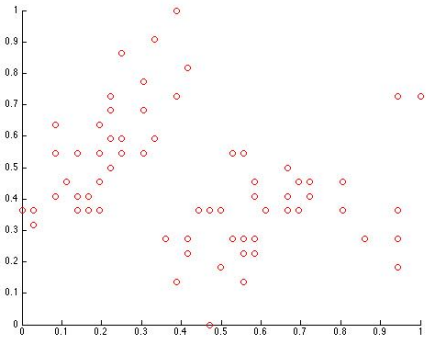


(b)

Problem 3. Perceptron as a classifier

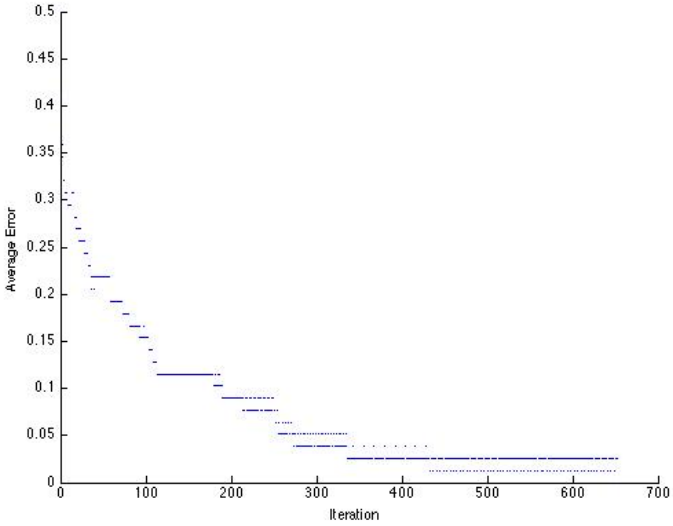
(a) Normalizing the data is useful because it shrinks the distances that have to be measured in the feature space while maintaining the relationships within the data, making computations easier. Gradient descent will converge much faster with normalization than without it.

(b) The classes are liberally separable because a decision boundary can be drawn with a straight line.



(c) See code in appendix.

(d) Average error rate: The average error rate of my perceptron is zero. It is designed to run over all the features, updating the weights, until it falls below an error threshold. I set the threshold to be below 0.001, so the average error rate it calculates eventually becomes zero:



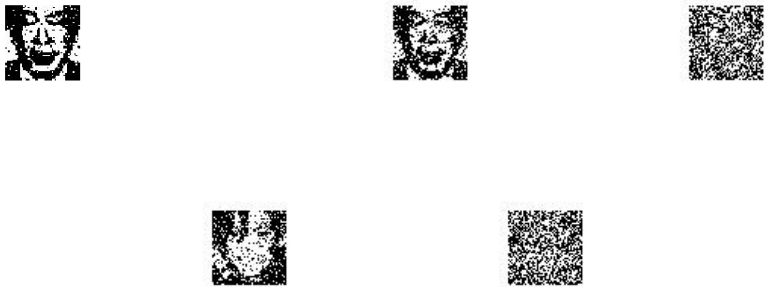
Problem 4. Hopfield Networks (b) (i) The number of nodes in the hop field network is $50^2 = 2500$ nodes. (ii) Type of connections: Fully connected and Symmetric. (iii) The update rule that minimizes the Energy function is very similar to the perceptron learning rule:

$$s_i \leftarrow \begin{cases} '1' & \text{if } \sum_j w_{ij} s_j \geq 0, \\ '0' & \text{otherwise.} \end{cases} \tag{4}$$

All of the thetas in this network are set to 0.

(c) The number of patterns that can be theoretically stored by the hop filed net is: $2500 * 0.138 = 345$ memories. The number of memories stored is dependent on neurons and their connections, and 0.138 is the limit on accurate recall between vectors and nodes.

(d) Example of corrupted images:



The original images:


(e) The following are recovered images:
Recovered image 1 from 10% corruption:

Recovered image 1 from 30% corruption:

Recovered image 1 from 80% corruption:

Recovered image 3 from 10% corruption:

Recovered image 3 from 30% corruption:

Recovered image 3 from 80% corruption:

SSE Table

Corruption Level	Image 1	Image 3
10%	0	113
30%	0	113
80%	1823	2001

(f) Yes, noisy images converged to a different memory than the original or expected memory. This happens when the hopfield network uses its update rule and winds up making the recalled image converge to a different final state than the expected state. A good analogy of how this happens is trying to recognize smudged text: A person (or hopfield network) can easily make the mistake of labeling a Q as an O if the lower right portion of the image is corrupted. Similarly, the more corrupted a face becomes, the more likely the hopfield network recalls an incorrect face. Often what can happen is the network causes a recollection to converge to a intermediary face, provided that such an intermediary is a local minima in the energy minimization function. This can cause a partially incorrect recollection instead of a completely incorrect one.

Listing 1: Perceptron Main

```
% Load in modified iris data
% raw_data = load('.\iris\iris_mod.txt');  % PC
raw_data = load('./iris/iris_mod.txt');  % Mac


X = raw_data(:,1:2);
Y = raw_data(:,3);

% Normalize each attribute (feature) of the data to th range [0,1]:
% X_n = (X - min(X))./(max(X) - min(X))
X_norm = (X - repmat(min(X, [], 1),[size(X,1),1]))./...
    repmat((max(X, [], 1) - min(X, [], 1)),[size(X,1),1]);


% Render a scatter plot using scatter(X,Y). Are the classes linearly
% separable? Why?
```

```
17 figure; hold on;
18 scatter(X_norm(:,1), X_norm(:,2), 'ro');
19
20
21 % Train a perceptron as a classifier using the delta learning rule on the
22 % training dataset available in iris_mod.txt. Include your source code in the
23 % % appendix.
24 % for i = 1:10
25 %     weight(:,i) = perceptron(X_norm, Y, .001, .001);
26 % end
27 % display(weight);
28 % avg_weight = mean(weight,2)
29 weight = perceptron(X_norm, Y, .001, .001);
30
31 % % x2 = (-thresh - w1*x1)/w2;
32 % x = 0:.1:1;
33 % y = (-1*weight(3)-weight(1).*x)./weight(2);
34 %
35 % plot(x,y);
36
37 % Classify the test data of file iris_test.txt an report the average error
38 % rate. The avg error is the number of misclassifid tst data points
39 % averaged over the number of test data points.
40
41
42
43
44 % clear raw_data X Y X_norm ans % Clean up workspace
```

Listing 2: perceptron

```
1  function [ weight ] = perceptron( observations, labels, learning_rate,...
2      error_thresh)
3  %PERCEPTRON Submission for problem 1 of section 2 of Gary' Neural Network
4  %course.
5  %   Inputs: observations, labels, learning_rate
6  %   Output: weight - a vector of weights + the threshold terms
7
8
9
10 N = size(observations,1);              % N = number of observations
11 num_input = size(observations,2);      % num_input = number of x inputs
12 num_output = size(labels,2);
13
14 % Weight vector = (Number of inputs + number of outputs (thresholds)) x 1.
15 weight = zeros(num_input+num_output,1);
16 weight(1:num_input) = rand(1,num_input);
```

```
17 weight(num_input+1:end) = -1*ones(num_output,1);
18 predictions = zeros(N,1);
19
20 figure; hold on;
21 err = calcError(predictions, labels);
22 plot(0, err);
23 xlabel('Iteration');
24 ylabel('Average Error');
25 i = 0;
26 while (err > error_thresh)
27
28     for feature = 1:N
29         predictions(feature) = predict(observations(feature,:), ...
30             weight(1:num_input), ...
31             weight(num_input+1:end));
32
33         weight = weight + learning_rate *...
34             (labels(feature)- predictions(feature)) ...
35             .* [observations(feature,:),ones(1,num_output)]';
36 %         display([weight' feature calcError(predictions, labels)]);
37     end
38
39     err = calcError(predictions, labels);
40     i = i + 1;
41     plot(i, err);
42
43
44 end
45 display('Avg error rate');
46 display(err);
47
48 end
```

Listing 3: predict

```
1  function [ prediction ] = predict(data, weight, thresh)
2  %UNTITLED4 Summary of this function goes here
3  %   Detailed explanation goes here
4  %   predict(observations(feature,:), weight)
5
6  if(data * weight + thresh > 0 )
7      prediction = 1;
8  else
9      prediction = 0;
10 end
11
12 end
```

Listing 4: calcError

```
function [ errorRate ] = calcError( predictions, labels )
%calcError Calculates and returns the error rate of the perceptron
%   Input: predictins, labels
%   Output: errorRate = average error rate

errorRate = abs(labels - predictions);
errorRate = sum(errorRate, 1)/size(labels,1);



end
```

Listing 5: Hopfield Network Main

```
% hw1_hopfieldnet

% Read images into Matlab using imread
% image_list = ls('.\faces\*.bmp'); % Get .bmp files in faces dir
image_list = dir('./faces/*.bmp'); % Mac version
num_images = size(image_list, 1); % Assign num images to var
images = zeros(50,50,num_images); % Initialize 3D matrix to store imgs
for i = 1:num_images
    images(:,:,i) = imread(['./faces/' image_list(i).name]); % imread imgs
end

% Explain with necessary text and mathematical statements the specification
% of the network you will use, videlicet:
% % Number of nodes in the hopfield network
% % ANS: 50^2 = 2500 Nodes
% % Types of connections
% % % Fully connected/Partially connected?
% % % Symmetric/Assymetric
% % ANS: The nodes will be partially connected and symmetric
% % Mathematical expressions for learning rule. Hint: Related to minimizing
% the Lyapunov energy function
% ANS: E = -1/2 Sum(w_ij*s_i*s_j, (i,j)) + Sum(Theta_i * s_i, i)


% Store the images in the Hopfield network according to the design
% specifications in the previous part. How many patterns can theoretically
% be stored by this hopfield network? Why?
% ANS: The capacity for memory in hopfield networks is 0.138 vectors
% (memories) per node (so it can store about 138 vectors for every 1000
% nodes). Thus, this network can store 345 memories.
hop_network = train_hopfield(images);
```

```
32
33
34 % Use the corrupt.m function provided in the util folder to corrupt any two
35 % input images. Try 10%, 30% and 80% corruption. Save these 6 corrupt
36 % images.
37 corr_imgs = zeros(size(images,1),size(images,2),6);
38 corr_lvls = [.80, .10, .30];
39 % for i = 1:6
40 %     corr_imgs(:,:,i) = corrupt(images(:,:,floor(i/4+1)),...
41 %         corr_lvls((mod(i,3) + 1)));
42 % end
43
44
45
46 % The corrupted images will serve as the test data to your trained Hopfield
47 % network. Write a function recover(hop_net, test_image, orig_image) that:
48 %% Reads the corrupted image into the Hopfield network
49 %% Runs it to convergence
50 %% Computs the Sum Squared Error (SSE) between the recovered image and the
51 %% original image.
52
53 corr_imgs(:,:,1) = corrupt(images(:,:,1),.1);
54 corr_imgs(:,:,2) = corrupt(images(:,:,1),.3);
55 corr_imgs(:,:,3) = corrupt(images(:,:,1),.8);
56
57 corr_imgs(:,:,4) = corrupt(images(:,:,3),.1);
58 corr_imgs(:,:,5) = corrupt(images(:,:,3),.3);
59 corr_imgs(:,:,6) = corrupt(images(:,:,3),.8);
60
61
62
63
64 [err10_im1 recim1] = recover(hop_network, corr_imgs(:,:,1), images(:,:,1));
65 [err30_im1 recim2] = recover(hop_network, corr_imgs(:,:,2), images(:,:,1));
66 [err80_im1 recim3] = recover(hop_network, corr_imgs(:,:,3), images(:,:,1));
67
68 [err10_im2 recim4] = recover(hop_network, corr_imgs(:,:,4), images(:,:,3));
69 [err30_im2 recim5] = recover(hop_network, corr_imgs(:,:,5), images(:,:,3));
70 [err80_im2 recim6] = recover(hop_network, corr_imgs(:,:,6), images(:,:,3));
71
72 im1 = [err10_im1 err30_im1 err80_im1];
73 im2 = [err10_im2 err30_im2 err80_im2];
74
75 % Do you encounter noisy image converging to a different memory than the
76 % one expected? If Yes, Why?
77
```

```
78 % Bonus: Implement the perceptron learning rule for Hopfield networks
79 % described in class. Write the function:
80 % hopfield_perceptron_train(image_list_fname, learning_rate,
81 % max_iterations) that iteratively trains the hopfield network on the
82 % images, starting from 0 wieghts. How well does this fare for the
83 % corrupted images in-terms of the SSE?
```

Listing 6: Update Hopfield

```
1  function [ new_img ] = update_hopfield(hop_net, new_pattern)
2  %UNTITLED2 Update a node in the hopfield network
3  %   Inputs: hop_net = the 2500x2500 weight matrix,
4  %           new_pattern = the new memory to try to recognize
5  %           ind = the index of the weight in the new pattern to update
6
7
8  l = size(new_pattern,1);
9  h = size(new_pattern,2);
10 new_img = zeros(size(new_pattern));
11
12 continue_flag = true;
13 while(continue_flag)
14     continue_flag = false;
15     rng('shuffle');
16     for i = randperm(size(hop_net,2)) % iterate in random order
17         new_img(i) = (reshape(new_pattern, 1, l*h) * hop_net(i, :)') >= 0;
18         if(new_img(i) ~= new_pattern(i))
19             continue_flag = true;
20         end
21     end
22     new_pattern = new_img;
23 end
24
25
26 end
```

Listing 7: Train Hopfield

```
1  function [ hop_net ] = train_hopfield(imgs)
2  %train_hopfield Trains the weights of the hopfield network
3  %   Input: imgs - 50x50xP matrix where P is the number of patters to train
4
5  P = size(imgs,3); % P = Number of patterns to store
6  l = size(imgs,2); % l = length of image, in this problem it is 50 px.
7
8  % There are 50^2 nodes in this network, so we need a 50^4 element matrix
9  % for the weights (they are symetrically connected, so we only need to
```

```
10 % calculate the upper right traingle).
11 hop_net = zeros(l*l,l*l);
12
13 % w = w0 + P*P' - I
14
15 for p = 1:P
16     img_tmp = reshape(imgs(:,:,p),1,size(imgs,1)*size(imgs,2));
17     img_tmp = img_tmp * 2 - 1;
18     hop_net = hop_net + img_tmp'*img_tmp - eye(l*l);
19 end
```

Listing 8: Recover

```
1  function [ SSE, recovered_img ] = recover( hop_net, test_image, orig_image )
2  %recover Inputs: hop_net, test_image, orig_image
3  %    Reads the corrupted image into the Hopfield network
4  %    Runs it to convergence
5  %    Computs the Sum Squared Error (SSE) between the recovered image and the
6  %    original image.
7
8
9  recovered_img = update_hopfield(hop_net, test_image);
10
11
12 % The square of 1 is 1, so this is effectively the SSE.
13 SSE = sum(sum(recovered_img ~= orig_image,1),2);
14
15
16
17
18
19 end
```

Submitted by Alex Rosengarten on January 29, 2015.