Alex Rosengarten
University of California, San Diego

Feb 26, 2015

HOMEWORK 3 — LaTeX

Problem 1. XOR

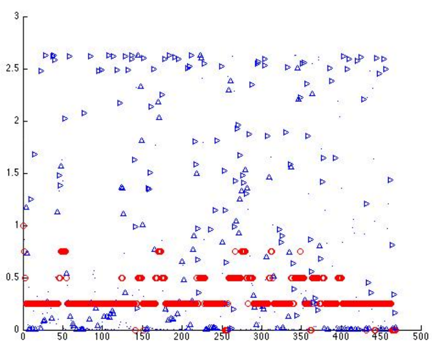
(a) The update rule for each layer is:

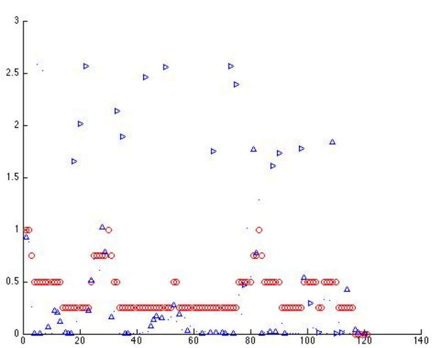
$\Delta w_{i,j}(t) := \gamma \delta_j(t) a_i(t) + \alpha \Delta w_{i,j}(t-1)$

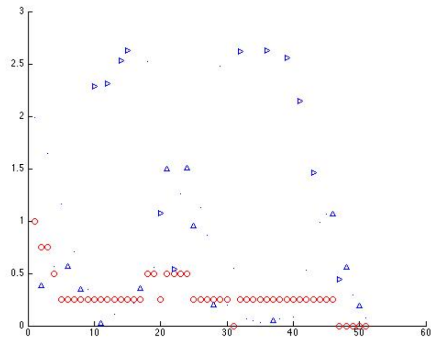
$w_{i,j} = w_{i,j} + \Delta w_{i,j}(t)$

(b)

The red circles are the error rates. The blue arrows are the errors at each output; the direction of the arrows are used to differentiate between the four possible inputs that could have produced that output.







(d) Mean: 2.0602e+03

Standard deviation: 1.4958e+03

Variance: 2.2375e+06

Epochs: 1341, 4720, 835, 964, 777 , 2239, 1230, 2388, 1412, 4696

Second iteration (used different learning rates for each layer):

Epochs: 362, 1275, 438, 540, 1323, 560, 932, 1216, 1129, 1161

Mean: 893.6000

Variance: 1.4319e+05

Standard Deviation: 378.3993

After I fixed my broken tanh (thank you Akshat!): Epochs: 861, 50, 38, 44, 46, 33, 321, 56, 463, 1848

Mean: 376

Standard Deviation: 584.2655

Variance: 3.4137e+05

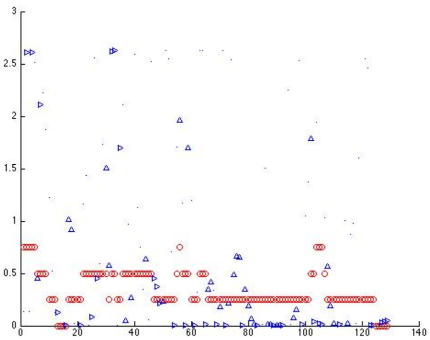
(e) In general, the number of epochs decreases as you increase the number of hidden units.

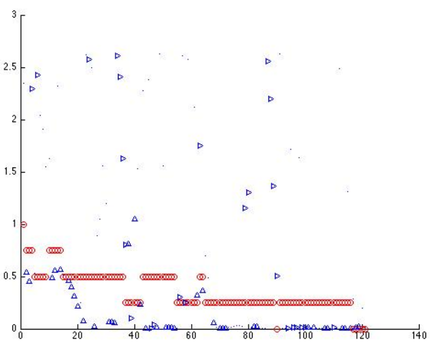
For 3 hidden units:

Epochs: 42, 58, 377, 62, 49, 118, 121, 129, 104, 58

Mean: 111.8000

Std: 98.8240



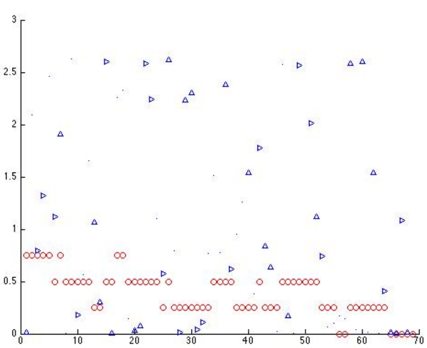


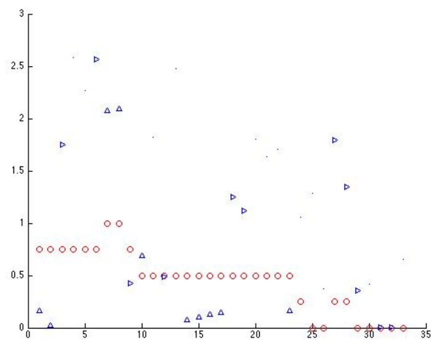
For 4 hidden units:

Epochs: 67, 94, 80, 85, 63, 73, 72, 67, 33, 69

Mean: 70.3000

Std: 16.1593





Problem 2. I converted the Y label vector to a 60,000 x 10 matrix so that I could get errors for each digit. I've included the average accuracy for each table.

I used two different tests to compare the output of the neural network to the label: First, a threshold value (if the probability was greater than 0.2, then the output becomes 1, otherwise it is a 0). Second, I chose the argument with the highest probability. These are the accuracies for each digit based on these techniques:

Accuracy Table: 100 hidden units, Sample 100 subsets at a time.

Theshold	Argmax
0.8279	0.6591
0.9334	0.7771
0.7914	0.8599
0.7593	0.8562
0.8066	0.8881
0.7957	0.9108
0.8727	0.9042
0.8912	0.9057
0.7643	0.9026
0.7956	0.8991

Accuracy Table: 50 hidden units, Sample 50 subsets at a time. Average accuracy: 0.8203, 0.8666.

Theshold	Argmax
0.8437	0.6798
0.9330	0.8111
0.7885	0.8502
0.8285	0.8817
0.8001	0.8951
0.7843	0.9107
0.8148	0.9042
0.8767	0.9311
0.7444	0.9026
0.7894	0.8991

Three examples of misclassified test instances and their corresponding true labels:

This table shows the counts of misclassifications: The rows indicate what the incorrect guesses were, the columns show what the incorrect guesses should have been.

	0	1	2	3	4	5	6	7	8	9
0	0	5	459	482	195	786	459	208	335	225
1	1	0	371	148	241	67	59	166	437	249
2	1	2	0	171	51	12	424	83	117	20
3	1	0	0	0	23	22	1	156	55	134
4	0	0	7	3	0	4	7	20	27	380
5	0	1	0	0	1	0	7	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	1	1	0	0	0	0	3	1
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

From the error rates and the table, it is clear that zero is the digit with the most errors. Zero is most commonly misclassified as six, one is most commonly misclassified as nine, and two is most commonly misclassified as seven.

This first example is a 5 misclassified as a 0:

This second example is a 7 misclassified as a 1:

The final example is a 9 misclassified as a 0:

Problem 3. Appendix

Listing 1: HW 3 Main file

```
% HW3 Main Script

```

```
% Problem 1: XOR
% Create a multi-layered neural net to implement XOR.
% User one hidden layer containing 2 units and an output layer containing
% 1 unit. Include a bias to each unit.
% Use backpropagation to train the weights.
% Your program should have two stopping criteria: it should stop when the
% error is below some threshold OR when a maximum number of epocks is
% reached, whatever comes first.


% PARAMETERS
N_inp = 2;
N_hid = 1;
N_out = 1;
N_bias = 1;
N_epochs = 5000;
numSuccess = 5;
alpha = 1;
momentum = 0.9;
K = 1; % Subset of training examples

% Create all combinations of inputs (plus bias)
X = ones(N_inp^2,N_inp + N_bias);
X(1:ceil(end/2), 1) = -1;
X(1:2:end, 2) = -1;
X(:,3) = 1;

Y = ones(1, 4);
Y([1, end]) = -1;

X_norm = zeros(size(X));
% Normalize the inputs and outputs
for j = 1:2 % exclude the bias
    x_std = std(X(:,j));
    x_mu  = mean(X(:,j));
    X_norm(:,j) = 1/x_std * (X(:,j) - x_mu);
end
X_norm(:,3) = 1;
disp('Starting NN');


funny_tanh = @(x) 1.7159*tanh((2*x)/3) ;%+ twisting_slope*x;
funny_tanh_prime = @(x) 1.14393*(1-tanh((2*x)/3).^2) ;%+ twisting_slope*x;

SSE = @(t, x, func) sum((t-double(func(x))).^2,2)/2;
SSE_prime = @(t,x, func, func_prime) sum((t-double(func(x)))*double(func_prime(x))*x,
```

```
[W_l1, W_l2, ep] = backprop_train(N_inp, N_hid, N_out, N_epochs, X_norm, Y, ...
    funny_tanh, funny_tanh_prime, numSuccess, alpha, momentum, K);

disp(ep);

% for p = 1:N_inp^2
%     A_h(:,p) = double(subs(funny_tanh,W_l1*X_norm(p,:)'));
%     A_o(:,p) = double(subs(funny_tanh,W_l2*[A_h(:,p); 1]));
%
% end

% tmp = [X(1,:)', X(1,:)', [A_h(:,1); 1]]';
% grad = ((Y(1) - A_o(1))*double(subs(diff(funny_tanh), W_l2*[A_h(:,p); 1])))*tmp;
% numgrad = (Y(1) - A_o(1)) * (Y(1) - A_o(1)) * (tmp - .0001);
%
% d = norm(grad-numgrad)/norm(grad+numgrad)

% grad_l1 = SSE_prime([Y; Y], W_l1*X(p,:)', funny_tanh, diff(funny_tanh));
% numgrad_l1 = computeNumericalGradient(@SSE, [Y; Y], A_h, 1.0e-4)
% diff_l1 = norm(grad_l1-numgrad_l1)/norm(grad_l1+numgrad_l1)
%
% grad_l2 = double(diff(SSE(Y, A_o)))
% numgrad_l2 = computeNumericalGradient(SSE, Y, A_o, 1.0e-4)
% diff_l2 = norm(grad_l2-numgrad_l2)/norm(grad_l2+numgrad_l2)
%


% Problem 2: MNIST

% % Import dataset
load('./MNSIT_mats/training_images.mat');
load('./MNSIT_mats/training_labels.mat');
load('./MNSIT_mats/test_images.mat');
load('./MNSIT_mats/test_labels.mat');


% PARAMETERS
N_inp = 784;
N_hid = 50;
N_out = 10;
N_bias = 1;
N_epochs = 5000;
numSuccess = 5;
alpha = 1; % Global learning rate
momentum = 0.9;
```

```
twisting_slope = 0.00;
K = 100;  % Subset of training examples
subset_size = 500;


funny_tanh = @(x) 1.7159*tanh((2*x)/3) ;%+ twisting_slope*x;
funny_tanh_prime = @(x) 1.14393*(1-tanh((2*x)/3).^2) ;%+ twisting_slope*x;


X_norm = [training_images, ones(size(training_images,1),1)];
X_subset = X_norm(1:subset_size,:);

Y =  zeros(size(training_labels,1), 10);
for j = 1:size(training_labels,1)
    Y(j,training_labels(j)+1) = 1;
end
Y_subset = Y(1:subset_size,:);


disp('Starting NN');
% [W_l1, W_l2, ep] = backprop_train_MNIST(N_inp, N_hid, N_out, N_epochs, X_norm, Y, ..
%      funny_tanh, funny_tanh_prime, numSuccess, alpha, momentum, K, true, true);

% TEST MNIST
X = [test_images, ones(size(test_images,1),1)];

Y =  zeros(size(test_labels,1), 10);
for j = 1:size(test_labels,1)
    Y(j,test_labels(j)+1) = 1;
end

thresh = 0.2;
testOut = zeros(N_out,size(test_images,1));
testOut2 = zeros(N_out,size(test_images,1));

for p = 1:size(test_images,1)
    A_h = funny_tanh(W_l1*X(p,:)');
    A_o(:,p) = funny_tanh(W_l2*[A_h; 1]);
    A_o(:,p) = exp(A_o(:,p))/sum(exp(A_o(:,p)));

    for i = 1:size(A_o,1)
        if(A_o(i,p) >= thresh)
            testOut(i,p) = 1;
        else
            testOut(i,p) = 0;
        end
```

```
        end

        testOut2(:,p) = 0;
        [~, currArgmax] = max(A_o(:,p));
        testOut2(currArgmax, p) = 1;



end
Yt = Y';
logical = testOut2 ~= Yt;
logical = logical - 2*Yt;

isVsShouldBe = zeros(10,10);

for i = 1:size(logical,2)
    shouldBeInd = find(logical(:,i) == -1);
    isInd = find(logical(:,i) == 1);
    isVsShouldBe(isInd, shouldBeInd) = isVsShouldBe(isInd, shouldBeInd) + 1;
end

max(isVsShouldBe)

err_rate = sum(testOut ~= Y',2)/size(Y,1);
err_rate2 = sum(testOut2 ~= Y',2)/size(Y,1);
acc1 = sum(testOut == Y', 2)/size(Y,1);
acc2 = sum(testOut2 == Y', 2)/size(Y,1);


[err_rate, err_rate2]
[acc1, acc2]
mean([acc1, acc2], 1)
```

Listing 2: Backprop train

```
function [W_l1, W_l2, ep] = backprop_train(N_inp, N_hid, N_out, N_epochs, ...
    X, Y, f, fprime, numSuccess, lrate, momentum, K)
%backprop_train Enter the size of the NN, the input data, error threshold,
% etc., this function will return the weights of the layers between
%the hidden layer and output layer.


% Create weight matricies
fan_in = 1/sqrt(N_inp+1);
W_l1 = fan_in - (-fan_in*rand(N_hid, N_inp+1)) + -fan_in;
W_l2 = fan_in - (-fan_in*rand(N_out, N_hid+1)) + -fan_in;

```

```
P = size(X,1);

% Create empty activation vectors
A_h = zeros(N_hid); % a^l = f(w^l*a^{l-1} + b^l)
A_o = zeros(N_out, P);


err_new = 0;
ep = 0;

dW_l2_new = zeros(size(W_l2));
dW_l1_new = zeros(size(W_l1));


lrate_l2 = .01 * lrate; %0.01 * sqrt(N_hid+1); %sqrt(N_hid+1) *0.5 * lrate
lrate_l1 = .5 * lrate;  %0.1  * sqrt(N_hid+1); %sqrt(N_inp+1) *0.99 * lrate
testOut = zeros(1,P);
figure; hold on
goodCount = 0;
while( goodCount < numSuccess && ep < N_epochs)

    err_subtotal = 0;

    for p = randsample(randperm(P), K)

        % Forward pass
        A_h = f(W_l1*X(p,:)');
        A_o(:,p) = f(W_l2*[A_h; 1]);


        testOut = sign(A_o);


        % Calculate the output error with teaching signal
        fpnet = fprime(W_l2*[A_h; 1]);
        D_o = (Y(p) - A_o(:,p)).*fpnet;

        err_subtotal = err_subtotal + sum(D_o.^2,1);


        % Update weights in layer 2 with momentum
        dW_l2_old = dW_l2_new;
        for j = 1:N_out
            dW_l2_new(j,:) = [lrate_l2*D_o(j).*[A_h; 1]]' + momentum*dW_l2_old(j,:);
        end
        W_l2 = W_l2 + dW_l2_new;
```

```

        % Calculate hidden layer error
        fpnet2 = [fprime(W_l1*X(p,:)')]';
        D_h = (D_o'*W_l2(:,1:end-1)).*fpnet2;


        % Change hidden layer weights
        dW_l1_old = dW_l1_new;
        for j = 1:N_hid
            dW_l1_new(j,:) = lrate_l1*D_h(j)*X(p,:) + momentum*dW_l1_old(j,:);
        end
        W_l1 = W_l1 + dW_l1_new;

    end

    A_o;
    err_rate = sum(testOut ~= Y)/size(Y,2)

    if(err_rate == 0)
        goodCount = goodCount + 1;
    elseif(goodCount > 0 & err_rate ~= 0)
        goodCount = goodCount - 1;
    end

    err_old = err_new;
    err_new = err_subtotal;

    ep = ep + 1

    styles = ['b^', 'b>', 'bv', 'b<'];
    plot(ep, err_rate, 'ro', ep, err_new, styles(mod(p,size(styles,2))+1));

end

if(goodCount >= numSuccess)
    ep = ep - numSuccess;
end


end
```

Listing 3: Backprop train

```
function [W_l1, W_l2, ep] = backprop_train_MNIST(N_inp, N_hid, N_out, N_epochs, ...
    X, Y, f, fprime, numSuccess, lrate, momentum, K, CE, SM)
%backprop_train Enter the size of the NN, the input data, error threshold,
% etc., this function will return the weights of the layers between
```

```
%the hidden layer and output layer.


% Create weight matricies
fan_in = 1/sqrt(N_inp+1);
W_l1 = fan_in - (-fan_in*rand(N_hid, N_inp+1)) + -fan_in;
W_l2 = fan_in - (-fan_in*rand(N_out, N_hid+1)) + -fan_in;

P = size(X,1);

% Create empty activation vectors
A_h = zeros(N_hid); % a^l = f(w^l*a^{l-1} + b^l)
A_o = zeros(N_out, P);


err_new = 0;
ep = 0;

dW_l2_new = zeros(size(W_l2));
dW_l1_new = zeros(size(W_l1));

thresh = 0.1;

lrate_l2 = .01 * lrate; %0.01 * sqrt(N_hid+1); %sqrt(N_hid+1) *0.5 * lrate
lrate_l1 = .1 * lrate;  %0.1  * sqrt(N_hid+1); %sqrt(N_inp+1) *0.99 * lrate
testOut = zeros(N_out,P);
testOut2 = zeros(N_out,P);
figure; hold on
goodCount = 0;
while( goodCount < numSuccess && ep < N_epochs)

    err_subtotal = 0;

    for p = randsample(randperm(P), K)

        % Forward pass
        A_h = f(W_l1*X(p,:)');
        A_o(:,p) = f(W_l2*[A_h; 1]);

        if(SM == true)
            A_o(:,p) = exp(A_o(:,p))/sum(exp(A_o(:,p)));
        end

        for i = 1:size(A_o,1)
            if(A_o(i,p) >= thresh)
               testOut(i,p) = 1;
```

```
            else
                testOut(i,p) = 0;
            end
        end

        testOut2(:,p) = 0;
        [~, currArgmax] = max(A_o(:,p));
        testOut2(currArgmax, p) = 1;


        % Calculate the output error with teaching signal
        if(CE == true)
            fpnet = 1;
        else
            fpnet = fprime(W_l2*[A_h; 1]);
        end

        D_o = (Y(p,:)' - A_o(:,p)).*fpnet;


        err_subtotal = err_subtotal + sum(D_o,1);


        % Update weights in layer 2 with momentum
        dW_l2_old = dW_l2_new;
        for j = 1:N_out
            dW_l2_new(j,:) = [lrate_l2*D_o(j).*[A_h; 1]]' + momentum*dW_l2_old(j,:);
        end

        W_l2 = W_l2 + dW_l2_new;

        % Calculate hidden layer error
        if(CE == true)
            fpnet2 = 1;
        else
            fpnet2 = [fprime(W_l1*X(p,:)')]';
        end
        D_h = (D_o'*W_l2(:,1:end-1)).*fpnet2;


        % Change hidden layer weights
        dW_l1_old = dW_l1_new;

        for j = 1:N_hid
            dW_l1_new(j,:) = lrate_l1*D_h(j)*X(p,:) + momentum*dW_l1_old(j,:);
        end
```

```

            W_l1 = W_l1 + dW_l1_new;

        end

        A_o;
        err_rate = sum(testOut ~= Y',2)/size(Y,1);
        err_rate2 = sum(testOut2 ~= Y',2)/size(Y,1);
        [err_rate, err_rate2]

        if(err_rate == 0)
            goodCount = goodCount + 1;
        elseif(goodCount > 0 & err_rate ~= 0)
            goodCount = goodCount - 1;
        end

        err_old = err_new;
        err_new = err_subtotal;

        ep = ep + 1

%       styles = ['b^', 'b>', 'bv', 'b<'];
        plot(ep, err_rate, 'r-', ep, err_rate2, 'b-');

end

if(goodCount >= numSuccess)
        ep = ep - numSuccess;
end


end
```

Submitted by Alex Rosengarten on Feb 26, 2015.