

Mat 128B Project 2: Backpropagation Neural Networks

Names: Nishad Mulay, Cole Warner, Michael-River Rose

Our team is organized in the following way: Michael-River focused on the first three parts of the project dealing with the MINST data base, plotting the digits, and implementing a neuron. Cole focused on parts four and five, creating and initializing the multilayer network. Nishad focused on the sixth and seventh parts, training the network on the test data and plotting the errors and their dependence on different parameters. Additionally, we made use of Github to track our progress, maintain a code base, and keep our work and resources straight. On the following page is a screen capture of our collaboration on Git.

The screenshot shows a GitHub repository page for 'MATH128B / ProjectTwo'. At the top, there are buttons for 'Watch', 'Star', and 'Fork', each with a count of 0. Below these are tabs for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. The main heading is 'Implementing neural networks' with an 'Edit' button. Below this, a summary bar shows '15 commits', '1 branch', '0 packages', '0 releases', and '3 contributors'. A table lists the repository's files and their commit history:

File	Commit Message	Time Ago
README.md	Update README.md	8 days ago
layerdependence.jpg	Add files via upload	2 hours ago
mnist_all.mat	Add files via upload	16 days ago
project2_p1	Create project2_p1	8 days ago
project2_p2.m	Add files via upload	8 days ago
project2_p3.m	Update project2_p3.m	8 days ago
project2_p4.m	Add files via upload	yesterday
project2_p5.m	Add files via upload	3 hours ago
project2_p6.m	Update project2_p6.m	3 hours ago
project2_p7.m	Add files via upload	2 hours ago

Below the table, the 'README.md' file is selected, showing its content:

ProjectTwo

Implementing neural networks

The elusive link to the website is [<https://academics.davidson.edu/math/chartier/Numerical/matlab.html>]. Then snag the script "mnist_all.mat", which should be the same one that Nishad uploaded here.

Make sure "mnist_all.mat" is in your matlab directory and then run it.

Now just type

```
digit = train0(1,:); digitImage = reshape(digit,28,28); image(rot90(flipud(digitImage),-1)), colormap(gray(256)), axis square tight off;
```

as suggested and you'll get an image of the 0-digit.

Everything else should be pretty straight forward from here.

i. MNIST Data Base

```
digit = train2(1,:);  
digitImage = reshape(digit,28,28);  
image(rot90(flipud(digitImage),-1))  
colormap(gray(256))  
axis square tight off
```

ii. Plot Digits

```
%Read digits from data base.  
%Type lines 3-6 in command window to produce a zero-image.  
digit = train0(1,:);  
digitimage = reshape(digit, 28, 28); %little i  
image(rot90(flipud(digitimage),-1));  
colormap(gray(256)), axis square tight off  
  
%Compute average digit (from 17a)  
%Plot them and compare with p. 180.  
T(1,:) = mean(train0);  
T(2,:) = mean(train1);  
T(3,:) = mean(train2);  
T(4,:) = mean(train3);  
T(5,:) = mean(train4);  
T(6,:) = mean(train5);  
T(7,:) = mean(train6);  
T(8,:) = mean(train7);  
T(9,:) = mean(train8);  
T(10,:) = mean(train9);  
  
for j = 1:10  
    subplot(2,5,j); %create 2x5 plot of 10 digits  
    i = T(j,:); %Loop through each T defined above  
    digitImage = reshape(i, 28, 28); %big I  
    image(rot90(flipud(digitImage),-1));  
    colormap(gray(256)), axis square tight off  
end
```



iii. A Neuron

```
format long      %compute with "Long" number of sig digits
```

```
% Examine relationships of connections and weights
```

```
C1 = [10; 20; 30];
```

```
w1 = [10 20 30];
```

```
Neuron(C1, w1)
```

```
% NET = 1400, OUT = 1
```

```
C2 = [1;1;1];
```

```
w2 = [2 2 2];
```

```
Neuron(C2, w2)
```

```
% NET = 6, OUT = 0.997527376843365
```

```
C3 = [.1;.1;.1];
```

```
w3 = [.2 .2 .2];
```

```
Neuron(C3, w3)
```

```
% NET = 0.060..., OUT = 0.514995501619410
```

```
% Given F = sigmoidal (logistic) function:
```

```
% As the values of NET decrease, so do the OUT values
```

```
function [OUT] = Neuron(C, w)
```

```
%INPUT: n input connections, "C"
```

```
%      n input weights "w"
```

```

NET = w * C
OUT = 1/(1+exp(-NET)); %Function we were given
end

```

iv. Multilayer Network

%Part 4 MultiLayer Network

```

function [O] = project2_p4(C1,w1)
                                % C1 is in format [x1,x2,x3,...,xn]
                                % w1 is in format [w11,w12,w13,...,w1n];
                                where each
O=C1;                           % w1i is a weight matrix
for i=1:length(w1)              % NETWORK = Xw where w is the input vector
    NETWORK= (w1(i).*O);        and X is the
                                O=1/(1+exp(-NETWORK)); % ith matrix contained in w1
                                O=O.'; % The output O=F(NETWORK)
end

```

v. Initializing the Network

%Part 5 Initializing the Network
%To initialize the network we assign random small numbers to each weight

```

n=3; %setting number of layers to 3
WMatrix=cell(1,n);
for i=1:n
    WMatrix{i}= rand([4 4]); %This assigns weights for each layer,
                                which
                                % will be random numbers between 0 and 1
end
WMatrix = cell2mat(WMatrix);
in=[1;2;3;4];
project2_p4(in,WMatrix)

```

vi. Training the Network

% part 6 - training the network

function percenterror = project2_p6(input, train, weight)

% input - cell array of input column vectors (C1,C2,etc.)

% train - cell array of output column vectors (train0, train1, etc.)

% weight - cell array of weight matrices (w1, w2, etc.)

layers = **length**(weight);

for i=1:**length**(input)

o = input(i);

out = cell(1, layers);

% forward pass

for j = 1:layers

net = o * weight(j); *% net = xw where x is the input
vector, C1,*

% and w is the weights in between

layers of

% neurons

o = 1./(1 + **exp**(-net)); *% o = f(net), this output vector will
be*

% the input of the next layer

out{j} = o;

end

*% backward pass - adjusts the hidden layers by propogating the
error*

% back and adjusting weights on the way

output = out{layers}; *% start with the last layer*

error = output - train(i); *% difference between output and
target*

% output

end

delta = output .* (1 - output) .* error; *% modified error*

updatedweight = out{layers-1}' * delta; *% iterated weight update*

```

                                % based on delta
weight(layers) = weight(layers) - updatedweight;

% for other hidden layers
for j= (layers-1):-1:2
    delta = (delta * weight(j+1)') * out{j} * (1 - out{j});
    updatedweight = out{j-1}' * delta;
    weight(j) = weight(j) - updatedweight;
end

out = cell2mat(out);
percenterror = ((1 - out)./ out)*100

end

```

vii. Dependence on Parameters

```

% part 7 - dependence on parameters
x = [0 1 2];

% ran the training program while changing the input and weights which
% affected the number of layers
% changed the output of the training program to percent error
a1 = project2_p6(C2,train1,w1);
a2 = project2_p6(C3,train1,w1);
a3 = project2_p6(C1,train1,w2);
a4 = project2_p6(C2,train1,w2);
a5 = project2_p6(C3,train1,w2);
a6 = project2_p6(C1,train1,w3);

figure
plot(x,a1,'LineWidth',2)
hold on
plot(x,a2,'LineWidth',2)
hold on
plot(x,a3,'LineWidth',2)
hold on
plot(x,a4,'LineWidth',2)

```

```
hold on
plot(x,a5,'LineWidth',2)
hold on
plot(x,a6,'LineWidth',2)
hold off
title('Dependence on Layers')
ylabel('Percent Error (%)')
```

