EEE 443 Neural Networks Homework 1 Report

# Question 1

# Question 2

## Part a

In the question following expression is expected to be implemented in four hidden layer neuron based network:

This expression can be expanded by the definition of XOR gate as such:

Due to duality first and the last terms can be expressed as product:

By expanding those terms the final result becomes:

In this question, activation function is set to be the unit step. Therefore, the function will be represented as u(x). In order to find the constraints, the truth tables must be established. The truth table associated with the first term is:

|  |  |  |  |
| --- | --- | --- | --- |
| X1 | X2 | X3 | out |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

Figure 1: Truth Table for

For this expression, constraints will be found in the following sense:

The truth table for the second expression is:

|  |  |  |  |
| --- | --- | --- | --- |
| X1 | X2 | X4 | out |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

Figure 2: Truth Table for

Constraints for this expression is:

The truth table of the third term is:

|  |  |  |  |
| --- | --- | --- | --- |
| X1 | X3 | X4 | Output |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Figure 3: Truth Table for

The final truth table is:

|  |  |  |  |
| --- | --- | --- | --- |
| X2 | X3 | X4 | Output |
|  | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

Figure 4: Truth Table for

Constraints of the final neuron are:

## Part b

Using the constraints in part a following weights and biases are selected:

For the first neuron:

For the second neuron:

For the third neuron:

For the fourth neuron:

For the output neuron:

The implementation of the neural network can be found in the appendix.  The results for the hidden layers and output layer neurons can be found below:

h1 =  0 0   0 0 1   1 0 0 0   0 0 0 0 0   0 0

h2 = 0   0 0 0   1 0 1 0   0 0 0 0 0   0 0 0

h3 = 0   0 0 0   0 0 0 0   0 0 0 1 0   0 0 1

h4 = 0   0 0 1   0 0 0 0   0 0 0 1 0   0 0 0

OUT =0   0 0 1   1 1 1 0   0 0 0 1 0   0 0 1

As it can be seen these outputs fit their corresponding truth table.

## Part c

In order to minimize error in the existence of Gaussian Noise, it is better to select biases in the middle of the their lower and upper boundaries in the constraints. Considering this notion, weights and biases are selected with respect to lesser error in the first place. If the biases are closer to border, output can be misclassified. The correctness of the aforementioned choices will be shown in part d as well.

## Part d

Success Rate of the model varies insignificantly around 91%. When other biases are applied using the same weights it is observed that success rate drops below 85%. In order to negate this notion in the part c is applied and the model became more effective.

# Appendix

## Code:

function Arda\_YUKSEL\_21601827\_hw1(question)

clc

close all

switch question

case '1'

disp('1')

%% question 1 is on the report

case '2'

disp('2')

%% question 2

% weights for the hidden layer

W1 = [-2 2 -2 0];

W2 = [-2 2 0 -2];

W3 = [2 0 3 1];

W4 = [0 -1 2 2];

%weight for the output neuron

Wout = [1 1 1 1],

% baises for the hidden layer

THETA\_1 = 1;

THETA\_2 = 1;

THETA\_3 = 5.5;

THETA\_4 = 3.5;

% bais for the output neuron

THETA = 0.5;

% x is the input vector consisting the binary vectors

disp('input:')

x = (decimalToBinaryVector(0:15))'

% hidden layers

h1 = unitStep(W1\*x-THETA\_1)

h2 = unitStep(W2\*x-THETA\_2)

h3 = unitStep(W3\*x-THETA\_3)

h4 = unitStep(W4\*x-THETA\_4)

% output layer

OUT = unitStep(Wout \*[h1; h2; h3; h4] -THETA)

% From the OUT, it can be seen that the model works in ideal case

%% part d

repeatedInput = repmat(x,1,25);

std = 0.2 % given in manual

%GNM is the Gaussian Noise Matrix

GNM = std\*randn(4,400)

noisedInput = repeatedInput + GNM

% for simplicity in coding Weights and Bais values are converted

% into matrix and vector forms respectively

WHidden = [W1; W2;W3;W4];

BaisHidden = [THETA\_1; THETA\_2; THETA\_3; THETA\_4];

% hidden layer output for the ideal input

IdealHidden = unitStep(WHidden\*repeatedInput-BaisHidden)

% hidden layer output for the noised input

NoisedHidden = unitStep(WHidden\*noisedInput-BaisHidden)

% output neurons for the ideal case and noised case

IdealOut = unitStep(Wout\*IdealHidden-THETA)

NoisedOut = unitStep(Wout\*NoisedHidden-THETA)

comparison = sum(IdealOut == NoisedOut)/4 % divide 4 due to number of rows

case '3'

disp('3')

%% question 3

load('assign1\_data1.mat')

%% part a

% in 200 images class changes

% two samples are used for correlation

firstSample = floor(200\* rand())+[1:200:5200]

secondSample = floor(200\* rand())+[1:200:5200]

while firstSample == secondSample

secondSample = floor(200\* rand())+[1:200:5200]

end

% these two will be used for within and across class correlation

imagesMatrix =[]

secondImagesMatrix = []

for i = 1:length(firstSample)

% image matrices are converted into column vectors to better

% calculate the correlations

firstImage = trainims(:,:, firstSample(i));

secondImage = trainims(:,:, secondSample(i));

columnVector = firstImage(:);

secondVector = secondImage(:);

imagesMatrix = [imagesMatrix columnVector];

secondImagesMatrix = [secondImagesMatrix secondVector];

% printing out the image

%figure

%image(trainims(:,:, sampleImages(i)))

end

disp('Across class Correlation:')

correlation\_matrix = corrcoef(double(imagesMatrix))

imagesc(correlation\_matrix)

disp('Within class Correlation:')

index = floor(26\*rand());%one of the letters will be used for within class

within\_class\_correlation = corrcoef(double(imagesMatrix(:,i)), double(secondImagesMatrix(:,i)))

% sample size for the within class can be incremented

%% part b

std = 0.1;

W = std\*randn(784,26);% 784 is the size of the column vector of each image

b = std\*randn(26,1);

nu = 0.2;

loss = [];

for i = 1:10000

index = randi([1 5200]);

im = double(trainims(:,:, index));

im\_vec = im(:)/255;%inputs are scaled

v = W'\*im\_vec - b;

y = sigmoid(v);

% gradient of the matrix will be calculated in following steps

% one hot is used for retrieving the vector associated with

% train labels

error = onehot(index,trainlbls)-y;

der = sigmoid(v).\*(1-sigmoid(v));%derivative of sigmoid

gradB = (error.\*der);%gradient of bais

gradW = (((error.\*der)').\*(im\_vec.\*ones(784,26)));%gradient of W

% bais and W are updated

W = W + nu \* gradW;

b = b + nu\* gradB;

loss\_stage = 0.5 \* error' \* error; % 0.5\* ||error||^2 loss of the cycle

loss = [loss loss\_stage];

end

%images of the weights

newW = reshape(W,28,28,26);

for i=1:26

weight = newW(:,:,i);

figure

hold on

imagesc(weight);

end

%testing results

positive = 0;

for i = 1:1300

im = double(testims(:,:, i));

im\_vec = im(:)/255;

v = W'\*im\_vec - b;

y = sigmoid(v);

[val,ind]=max(y);%ind will be used in understanding the output letter

if(testlbls(i)==ind)

positive = positive +1;%for matches the positive value is incremented

end

end

fprintf('For nu = 0.2(optimal) the success rate is = %f\n',positive/1300\*100);

%% part c and d

%high

%values are updated and resetted

nuHigh= 0.95;

WHigh = std\*randn(784,26);

bHigh = std\*randn(26,1);

lossHigh = [];

for i = 1:10000

index = randi([1 5200]);

im = double(trainims(:,:, index));

im\_vec = im(:)/255;

v = WHigh'\*im\_vec - bHigh;

y = sigmoid(v);

error = onehot(index,trainlbls)-y;

der = sigmoid(v).\*(1-sigmoid(v));%derivative of sigmoid

gradB = (error.\*der);

gradW = (((error.\*der)').\*(im\_vec.\*ones(784,26)));

WHigh = WHigh + nuHigh \* gradW;

bHigh = bHigh + nuHigh\* gradB;

loss\_stage = 0.5 \* error' \* error; % 0.5\* ||error||^2 loss of the cycle

lossHigh = [lossHigh loss\_stage];

end

positive = 0;

for i = 1:1300

im = double(testims(:,:, i));

im\_vec = im(:)/255;

v = WHigh'\*im\_vec - bHigh;

y = sigmoid(v);

[val,ind]=max(y);

if(testlbls(i)==ind)

positive = positive +1;

end

end

fprintf('For nu = 0.95(high) the success rate is = %f\n',positive/1300\*100);

%low

%values are updated and resetted

nuLow= 0.95;

WLow = std\*randn(784,26);

bLow = std\*randn(26,1);

lossLow = [];

for i = 1:10000

index = randi([1 5200]);

im = double(trainims(:,:, index));

im\_vec = im(:)/255;

v = WLow'\*im\_vec - bLow;

y = sigmoid(v);

error = onehot(index,trainlbls)-y;

der = sigmoid(v).\*(1-sigmoid(v));%derivative of sigmoid

gradB = (error.\*der);

gradW = (((error.\*der)').\*(im\_vec.\*ones(784,26)));

WLow = WLow + nuLow \* gradW;

bLow = bHigh + nuLow\* gradB;

loss\_stage = 0.5 \* error' \* error; % 0.5\* ||error||^2 loss of the cycle

lossLow = [lossLow loss\_stage];

end

positive = 0;

for i = 1:1300

im = double(testims(:,:, i));

im\_vec = im(:)/255;

v = WLow'\*im\_vec - bLow;

y = sigmoid(v);

[val,ind]=max(y);

if(testlbls(i)==ind)

positive = positive +1;

end

end

fprintf('For nu = 0.02(low) the success rate is = %f\n',positive/1300\*100);

%data visualization

axis = 1:10000;

figure;

plot(axis,loss);

hold on;

plot(axis,lossHigh);

plot(axis,lossLow);

title('Loss Function-Learning Rates');

legend('learning rate 0.2', 'high learning rate 0.95', 'low learning rate 0.02');

xlabel('Cycle');

ylabel('Loss Function');

end

end

function t = onehot(x,label)

temp = zeros(26,1);

index = label(x);

temp(index,1) = 1;

t = temp;

end

function y = unitStep(x)

y = x >= 0;

end

function y = sigmoid(x)

y = 1./(1+exp(-x));

end