EE4305 Assignment 1

Adil Bin Azmoon

A0155858M

Group 3

Q1a.

f(x,y) = (1-x)2+100(y-x2)2

Therefore:

fx = 2(x-1)-400(y-x2)

fy = 200y-200x2

Using Steepest Descent Method,

w(k+1) = w(k) + ηg(k), where η = 0.001 and g(k) = ∇f(w(k))

Using the following Python script,

**n = 0.001**

**def derivX(x,y):**

**answer = (2\*(x-1)) - (400\*x\*(y-(x\*\*2)))**

**return(answer)**

**def derivY(x,y):**

**answer = 200\*(y-(x\*\*2))**

**return(answer)**

**def deltaX(x,y):**

**answer = x - (n\*derivX(x,y))**

**return answer**

**def deltaY(x,y):**

**answer = y - (n\*derivY(x,y))**

**return answer**

**def f(x,y):**

**answer = ((1-x)\*\*2) + 100\*((y-(x\*\*2))\*\*2)**

**return answer**

**def steepestdescent(x,y,n):**

**if (f(x,y)<.1):**

**return 0**

**else:**

**newX = deltaX(x,y)**

**newY = deltaY(x,y)**

**print("Wave :" + str(n))**

**print("x: " + str(newX))**

**print("y: " + str(newY))**

**print("f(x,y): " + str(f(newX, newY)))**

**return steepestdescent(newX,newY,(n+1))**

**steepestdescent(0,0.5,0)**

Taking f(x,y) < 0.1 as approaching 0,

This happens at iteration 1040, where

x = 0.6842420004628382

y = 0.46668078428343124

f(x,y) = 0.09993001755394404

if a larger learning rate of 0.2 is used, the weights would be adjusted to drastically after each iteration, causing it to have to be adjusted back in the opposite direction after each iteration.

Q1b.

fyy = 200

fxx = 1200x2 – 400y + 2

fxy = fyx = -400x

Using Newton’s method,

∆w(k) = -H-1(k)g(k), where H-1(k) is the inverse Hessian Matrix of f(x,y) and g(k) = ∇f(w(k))

Using the following Python script,

**def derivX(x,y):**

**answer = (2\*(x-1)) - (400\*x\*(y-(x\*\*2)))**

**return(answer)**

**def derivY(x,y):**

**answer = 200\*(y-(x\*\*2))**

**return(answer)**

**def derivXX(x,y):**

**answer = 2-(400\*(y-(3\*(x\*\*2))))**

**return(answer)**

**def derivYY(x,y):**

**answer = 200**

**return(answer)**

**def derivXY(x,y):**

**answer = -400\*x**

**return(answer)**

**def derivYX(x,y):**

**answer = -400\*x**

**return(answer)**

**def inverseHX(x,y):**

**multiplier = -1/((derivXX(x,y)\*derivYY(x,y))-(derivXY(x,y)\*derivYX(x,y)))**

**answer = multiplier \* ((derivYY(x,y)\*derivX(x,y)) + (-(derivXY(x,y))\*derivY(x,y)))**

**print("DeltaX: " + str(answer))**

**return answer**

**def inverseHY(x,y):**

**multiplier = -1/((derivXX(x,y)\*derivYY(x,y))-(derivXY(x,y)\*derivYX(x,y)))**

**answer = multiplier \* ((-(derivYX(x,y))\*derivX(x,y)) + (derivXX(x,y)\*derivY(x,y)))**

**print("DeltaY: " + str(answer))**

**return answer**

**def f(x,y):**

**answer = ((1-x)\*\*2) + 100\*((y-(x\*\*2))\*\*2)**

**return answer**

**def recursion(x,y,n):**

**if (f(x,y)<0.1):**

**return 0**

**else:**

**print("Wave :" + str(n))**

**newX = x + inverseHX(x,y)**

**newY = y + inverseHY(x,y)**

**print("x: " + str(newX))**

**print("y: " + str(newY))**

**print("f(x,y): " + str(f(newX, newY)))**

**print()**

**return recursion(newX,newY,(n+1))**

**recursion(0,0.5,1)**

Taking f(x,y) < 0.1 as approaching 0,

at iteration 3, where

x = 0.9799025478636456

y = 0.9602089927935581

f(x,y) = 0.0004039075823841144

Q2a.

**Matlab code:**

input = [];

output = [];

test = [];

test\_output = [];

for i = -1: 0.05: 1

input = [input, i];

output\_val = 1.2\*sin(pi\*i)-cos(2.4\*pi\*i);

output = [output, output\_val];

end

for j = -1: 0.01: 1

test = [test, j];

test\_output\_val = 1.2\*sin(pi\*j)-cos(2.4\*pi\*j);

test\_output = [test\_output, test\_output\_val];

end

input\_c = num2cell(input,1);

output\_c = num2cell(output,1);

for n = 1: 10

net = fitnet(n);

[net,y,e,pf] = adapt(net, input\_c, output\_c);

test\_results(n,:) = net(test); % predictions on training set

resultsOfThree(n,:) = net(3);

resultsOfNThree(n,:) = net(-3);

end

net = fitnet(20);

[net,y,e,pf] = adapt(net, input\_c, output\_c);

test\_results(11,:) = net(test); % predictions on training set

resultsOfThree(11,:) = net(3);

resultsOfNThree(11,:) = net(-3);

net = fitnet(50);

[net,y,e,pf] = adapt(net, input\_c, output\_c);

test\_results(12,:) = net(test); % predictions on training set

resultsOfThree(12,:) = net(3);

resultsOfNThree(12,:) = net(-3);

Using sequential training, the MLP was unable to learn the pattern for the function, especially considering that the input range of -1 to 1 is significantly less than the period of the function. For all values of n, the MLP was inaccurate. However, it should be noted that from n = [1,4], the ouput of the MLP was within the maximum and minimum range of the expected output, while further neurons beyond that pushed the values beyond that range. In this case, it was found that n=4 was the closest fitting MLP.

Expected values of y when x = 3 and x = -3:

y(3) = y(-3) = 0.809

Actual values were extremely inacurrate.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | 3 | -3 |
| Number of hidden neurons | 1 | 1.226112 | -1.13678 |
| 2 | -0.85915 | 0.132506 |
| 3 | 3.998924 | -4.47331 |
| 4 | 1.210427 | -2.92484 |
| 5 | -0.56946 | -4.28121 |
| 6 | 0.949564 | -2.80311 |
| 7 | 3.833746 | -9.55268 |
| 8 | 3.709962 | -1.71787 |
| 9 | 6.272822 | -6.23805 |
| 10 | 5.409101 | -4.0087 |
| 20 | 0.951481 | 0.814356 |
| 50 | -4.25241 | 5.521925 |

Q2b.

**Matlab code:**

input = [];

output = [];

test = [];

for i = -1: 0.05: 1

input = [input, i];

output\_val = 1.2\*sin(pi\*i)-cos(2.4\*pi\*i);

output = [output, output\_val];

end

for j = -1: 0.01: 1

test = [test, j];

end

input\_c = num2cell(input,1);

output\_c = num2cell(output,1);

for n = 1: 10

net = fitnet(n, 'trainlm');

net = train(net, input, output);

test\_results(n,:) = net(test); % predictions on training set

resultsOfThree(n,:) = net(3);

resultsOfNThree(n,:) = net(-3);

end

net = fitnet(20, 'trainlm');

net = train(net, input, output);

test\_results(11,:) = net(test); % predictions on training set

resultsOfThree(11,:) = net(3);

resultsOfNThree(11,:) = net(-3);

net = fitnet(50, 'trainlm');

net = train(net, input, output);

test\_results(12,:) = net(test); % predictions on training set

resultsOfThree(12,:) = net(3);

resultsOfNThree(12,:) = net(-3);

Using batch training yielded greater performance for the MLP. The MLP was underfitting for n=[1,5], proper fitting for n=[6,10], slightly overfitting at n=20 and extremely overfitting at n=50.

Expected values of y when x = 3 and x = -3:

y(3) = y(-3) = 0.809

Actual values were extremely inacurrate.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | 3 | -3 |
| Number of hidden neurons | 1 | 0.896423 | -2.19306 |
| 2 | 0.879997 | -3.11474 |
| 3 | -0.51688 | -1.52952 |
| 4 | -0.54531 | 2.501431 |
| 5 | -0.91907 | 2.051256 |
| 6 | 0.039369 | 0.184814 |
| 7 | 2.089936 | 0.536384 |
| 8 | 1.147581 | 0.46162 |
| 9 | 0.367003 | -0.42733 |
| 10 | -5.46086 | 0.11984 |
| 20 | 0.156452 | 0.817889 |
| 50 | -0.57575 | 2.41682 |

Q2c.

**Matlab code:**

input = [];

output = [];

test = [];

for i = -1: 0.05: 1

input = [input, i];

output\_val = 1.2\*sin(pi\*i)-cos(2.4\*pi\*i);

output = [output, output\_val];

end

for j = -1: 0.01: 1

test = [test, j];

end

input\_c = num2cell(input,1);

output\_c = num2cell(output,1);

for n = 1: 10

net = fitnet(n, 'trainbr');

net = train(net, input, output);

test\_results(n,:) = net(test); % predictions on training set

resultsOfThree(n,:) = net(3);

resultsOfNThree(n,:) = net(-3);

end

net = fitnet(20, 'trainbr');

net = train(net, input, output);

test\_results(11,:) = net(test); % predictions on training set

resultsOfThree(11,:) = net(3);

resultsOfNThree(11,:) = net(-3);

net = fitnet(50, 'trainbr');

net = train(net, input, output);

test\_results(12,:) = net(test); % predictions on training set

resultsOfThree(12,:) = net(3);

resultsOfNThree(12,:) = net(-3);

Using batch training yielded greater performance for the MLP. The MLP was underfitting for n=[1,2], slightly underfitting at n=[3,4], proper fitting for n=[5,50] with no overfitting on any number of hidden neurons.

Expected values of y when x = 3 and x = -3:

y(3) = y(-3) = 0.809

Actual values were inacurrate.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | 3 | -3 |
| Number of hidden neurons | 1 | 0.829195 | -1.50635 |
| 2 | -0.86074 | -0.87995 |
| 3 | -3.16636 | 6.302535 |
| 4 | -0.5546 | 5.286107 |
| 5 | -0.35805 | 15.05729 |
| 6 | 0.820677 | 3.533887 |
| 7 | 0.82393 | 3.455438 |
| 8 | 0.790011 | 3.29064 |
| 9 | 0.73441 | 3.247023 |
| 10 | 0.81388 | 3.173101 |
| 20 | 0.525707 | 2.602642 |
| 50 | 1.262539 | 3.079869 |

Q3a.

**Matlab code:**

train\_folder = dir('group\_3/train');

val\_folder = dir('group\_3/val');

training\_namelist = {train\_folder.name}.';

training\_namelist = training\_namelist(3:503);

val\_namelist = {val\_folder.name}.';

val\_namelist = val\_namelist(3:169);

train\_images = [];

val\_images = [];

% Populate training images matrix

for i = 1: size(training\_namelist)

name = "group\_3/train/" + training\_namelist{i};

name = convertStringsToChars(name);

tmp = strsplit(name, {'\_', '.'});

training\_labels(i)= str2num(tmp{3});

I = imread(name);

V=I(:);

train\_images = [train\_images,V];

end

% Convert to double for matrix multiplication

train\_images = double(train\_images);

% initialise weights at all 0

w = zeros(1, 65536);

learning\_rate = 1;

%Iterate through 500 images

for j = 1:size(training\_namelist)

% Calculate output

output = w\*train\_images(:,j);

% Hard limiter

if (output>0)

output = 1;

else

output = 0;

end

% Update weights if wrong class

if (output ~= training\_labels(j))

for k = 1:65536

w(k) = w(k) + (learning\_rate\*(training\_labels(j)-output)\*train\_images(k,j));

end

end

end

% VALIDATION STAGE

% Populate validation images matrix

for i = 1: size(val\_namelist)

name = "group\_3/val/" + val\_namelist{i};

name = convertStringsToChars(name);

tmp = strsplit(name, {'\_', '.'});

validation\_labels(i)= str2num(tmp{3});

I = imread(name);

V=I(:);

val\_images = [val\_images,V];

end

% Convert to double for matrix multiplication

val\_images = double(val\_images);

predictions = [];

passes = 0;

for j=1:167

predictions(j) = w\*val\_images(:,j);

% Hard limiter

if (predictions(j)>0)

predictions(j) = 1;

else

predictions(j) = 0;

end

if predictions(j)==validation\_labels(j)

passes = passes + 1;

end

end

accuracy = passes/167;

Classification of the validation set yielded 109 passes out of 167 images for a total accuracy of 0.6527. Rosenblatt’s perceptron performed moderately. However, this may not reflect actual performance, as the results are due to a sum total of pixel values multiplied with their respective weights. The single perceptron does not, in fact, learn patterns and should thus be unable to differentiate between the two classes effectively.

Q3b.

**Matlab code:**

train\_folder = dir('group\_3/train');

val\_folder = dir('group\_3/val');

training\_namelist = {train\_folder.name}.';

training\_namelist = training\_namelist(3:503);

val\_namelist = {val\_folder.name}.';

val\_namelist = val\_namelist(3:169);

train\_images = [];

val\_images = [];

% Populate training images matrix

for i = 1: size(training\_namelist)

name = "group\_3/train/" + training\_namelist{i};

name = convertStringsToChars(name);

tmp = strsplit(name, {'\_', '.'});

training\_labels(i)= str2num(tmp{3});

I = imread(name);

I = imresize(I, [32 32]);

V=I(:);

train\_images = [train\_images,V];

end

% Convert to double for matrix multiplication

train\_images = double(train\_images);

% initialise weights at all 0

w = zeros(1, 1024);

learning\_rate = .001;

%Iterate through 500 images

for j = 1:size(training\_namelist)

% Calculate output

output = w\*train\_images(:,j);

% Hard limiter

if (output>0)

output = 1;

else

output = 0;

end

% Update weights if wrong class

if (output ~= training\_labels(j))

for k = 1:size(V)

w(k) = w(k) + (learning\_rate\*(training\_labels(j)-output)\*train\_images(k,j));

end

end

end

% VALIDATION STAGE

% Populate validation images matrix

for i = 1: size(val\_namelist)

name = "group\_3/val/" + val\_namelist{i};

name = convertStringsToChars(name);

tmp = strsplit(name, {'\_', '.'});

validation\_labels(i)= str2num(tmp{3});

I = imread(name);

I = imresize(I, [32 32]);

V=I(:);

val\_images = [val\_images,V];

end

% Convert to double for matrix multiplication

val\_images = double(val\_images);

predictions = [];

passes = 0;

for j=1:167

predictions(j) = w\*val\_images(:,j);

% Hard limiter

if (predictions(j)>0)

predictions(j) = 1;

else

predictions(j) = 0;

end

if predictions(j)==validation\_labels(j)

passes = passes + 1;

end

end

accuracy = passes/167;

There was no change in performance at all when the images were downsized to 128x128 and 64x64. When downsized to 32x32, there was only a marginal improvement of 110 passes.

Q3c.

**Matlab code:**

train\_folder = dir('group\_3/train');

val\_folder = dir('group\_3/val');

training\_namelist = {train\_folder.name}.';

training\_namelist = training\_namelist(3:503);

val\_namelist = {val\_folder.name}.';

val\_namelist = val\_namelist(3:169);

train\_images = [];

val\_images = [];

% Populate training images matrix

for i = 1: size(training\_namelist)

name = "group\_3/train/" + training\_namelist{i};

name = convertStringsToChars(name);

tmp = strsplit(name, {'\_', '.'});

training\_labels(i)= str2num(tmp{3});

I = imread(name);

V=I(:);

train\_images = [train\_images,V];

end

% Convert to double for matrix multiplication

train\_images = double(train\_images);

net = patternnet(64);

net = train(net, train\_images, training\_labels);

% VALIDATION STAGE

% Populate validation images matrix

for i = 1: size(val\_namelist)

name = "group\_3/val/" + val\_namelist{i};

name = convertStringsToChars(name);

tmp = strsplit(name, {'\_', '.'});

validation\_labels(i)= str2num(tmp{3});

I = imread(name);

V=I(:);

val\_images = [val\_images,V];

end

% Convert to double for matrix multiplication

val\_images = double(val\_images);

pred\_train = round(net(train\_images));

acc\_train = 1 - mean(abs(pred\_train - training\_labels));

pred\_val = round(net(val\_images));

acc\_val = 1 - mean(abs(pred\_val - validation\_labels));

|  |  |
| --- | --- |
| Classification accuracy | |
| Training set | Validation set |
| 0.82235529 | 0.71257485 |

Batch mode training yielded moderately accurate results which were 5% higher than that of the perceptron in 3a.

Q3d.

**Matlab code:**

train\_folder = dir('group\_3/train');

val\_folder = dir('group\_3/val');

training\_namelist = {train\_folder.name}.';

training\_namelist = training\_namelist(3:503);

val\_namelist = {val\_folder.name}.';

val\_namelist = val\_namelist(3:169);

train\_images = [];

val\_images = [];

epoch = 10;

% Populate training images matrix

for i = 1: size(training\_namelist)

name = "group\_3/train/" + training\_namelist{i};

name = convertStringsToChars(name);

tmp = strsplit(name, {'\_', '.'});

training\_labels(i)= str2num(tmp{3});

I = imread(name);

V=I(:);

train\_images = [train\_images,V];

end

% Convert to double for matrix multiplication

train\_images = double(train\_images);

net = patternnet(64);

for i=1:epoch

net = adapt(net, train\_images, training\_labels);

end

% VALIDATION STAGE

% Populate validation images matrix

for i = 1: size(val\_namelist)

name = "group\_3/val/" + val\_namelist{i};

name = convertStringsToChars(name);

tmp = strsplit(name, {'\_', '.'});

validation\_labels(i)= str2num(tmp{3});

I = imread(name);

V=I(:);

val\_images = [val\_images,V];

end

% Convert to double for matrix multiplication

val\_images = double(val\_images);

pred\_train = round(net(train\_images));

acc\_train = 1 - mean(abs(pred\_train - training\_labels));

pred\_val = round(net(val\_images));

acc\_val = 1 - mean(abs(pred\_val - validation\_labels));

Batch mode training yielded moderate results which were lower than that of batch mode training. However, these results are only from 1 epoch of incremental training. High number of epochs yielded a significant increase in classification accuracy in the training set. However, this is only due to overtraining, as there was no significant improvement in performance in the validation set.

|  |  |  |
| --- | --- | --- |
|  | Classification accuracy | |
|  | Training set | Validation set |
| 1 | 0.71856287 | 0.706586826 |
| 2 | 0.62075848 | 0.562874251 |
| 3 | 0.65868263 | 0.604790419 |
| 4 | 0.64271457 | 0.592814371 |
| 5 | 0.78642715 | 0.730538922 |
| 6 | 0.78842315 | 0.71257485 |
| 7 | 0.84431138 | 0.706586826 |
| 8 | 0.65868263 | 0.604790419 |
| 9 | 0.80439122 | 0.676646707 |
| 10 | 0.79041916 | 0.688622754 |
| 20 | 0.9241517 | 0.71257485 |
| 50 | 0.99001996 | 0.724550898 |